ملخصات شوم

نظريات و مسائل

فى

# البرمجة بلغة الباسكال

تأليف


بايرون س. جوتفريد Ph. D.

أستاذ الهندسة الصناعية

هندسة إدارة النظم وبحوث العمليات

جامعة بيتسبرج


ترجمة ومراجعة


دكتور/ **سرور على إبراهيم سرور**

كلية الاقتصاد والإدارة

جامعة الملك سعود – فرع القصيم


تقديم


دكتور / **عبد الله بن عبد الله العبيد**

عميد كلية الاقتصاد والإدارة

جامعة الملك سعود – فرع القصيم



**الدار الدولية للنشر والتوزيع**

# البرمجة بلغة الباسكال

Theory and problems of

PROGRAMMING WITH PASCAL

by

BYRON S. GOTTFRIED



**الدار الدولية للنشر والتوزيع**

ص. ب ٥٥٩٩ هليوبوليس غرب – القاهرة

ت : ٢٥٨٢٨٨٧

تلكس : ٢٠٠٧٠ PBCRB UN

فاكس : ٢٩١٨٠٥٩ / ٠٠٢٠٢

بسم الله الرحمن الرحيم

# تقديم

الحمد لله رب العالمين ، والصلاة والسلام على أشرف المرسلين ، سيدنا محمد ، وعلى آله وصحبه ، ومن تبعهم بإحسان إلى يوم الدين .

وبعـــد :

فما من شك في أن تطوير القدرات والمهارات لاستخدام جهاز الكمبيوتر أصبح من الأمور الضرورية في هذا العصر . فهذا الجهاز توغل في حياتنا ، حتى أصبح مؤثراً تأثيراً مباشرا أو غير مباشر على كل فرد منا .

فمما هو جدير بالذكر أن هناك لغات برمجة خاصة تستخدم مع الكمبيوتر . ومن أهم هذه اللغات في وقتنا الحالي « لغة البسكال » التي سميت باسم عالم الرياضيات والفيزياء الفرنسي « بليز بسكال » تكريما له ، حيث إنه أعد عام ١٦٤٥ م آله حاسبة قادرة على جمع وطرح الأعداد . وقد كان لإعداد هذه الآلة دوى كبير في الأوساط العلمية التي لم تكن تتوقعها ، مما جعل الناس لايتقبلونها بيسر في ذلك الوقت . وبعد ظهور الكمبيوتر وانتشاره ؛ لاقت لغة البسكال انتشارا سريعا ، حيث إنها أعدت لعمل برمجة مرتبة ومنظمة للكمبيوتر .

وقد وقع الاختيار على كتاب « البرمجة بلغة البسكال » لمؤلفه الدكتور / بيرون جوتفريد ، أستاذ الهندسة الصناعية بجامعة بتسبرج بأمريكا ، لأهمية المادة العلمية الموجودة فيه ، وسهولة عرضه لها . وحتى تتاح الفرصة للافادة من هذا الكتاب على وجه أفضل ، اتجه الرأى إلى ترجمته ليستخدم كأساس للتدريس في الدورات التدريبية التي يقدمها مركز البحوث وتنمية الموارد البشرية بكلية الاقتصـاد والإدارة . وقام بترجمته الدكتور / سرور على إبراهيم سرور ، الأستاذ المشارك بقسم الأساليب الكمية بالكلية ، وقام بمراجعته الأستاذ / عاصم أحمد الحماحمي ، المشرف على معمل الكمبيوتر بالكلية .

هذا .. ونرجو أن يثيبها الله خير الثواب على ماقاما به من جهد مشكور في إعداد هـذا الكـتاب . كما نأمل أن يكونا قد قدما إلى مكتبتنا العربية كتابا جديدا في بابه ، يحقق الغرض المرجـو ، ويحقق الهدف المقصود .

والله من وراء القصد ،

**د . عبد الله بن عبد الله العبيد**
عميد كلية الاقتصاد والإدارة
جامعة الملك سعود – فرع القصيم

بسم الله الرحمن الرحيم

## مقدمة المترجم

أن أساس لغة البسكال هي لغة الجول ٦٠ ( Algol 60 ) . وقد أعدت لغة الجول ٦٠ في أصلها كلغة للتفاهم بين علماء الرياضيّات . ومع وجود الكمبيوتر ، وجد أنه من الممكن استخدام هذه اللغة كإحدى لغات البرمجة للكمبيوتر . وهذا يبين لنا لماذا لم تلق هذه اللغة نجاحا كبيرا كاحدى لغات البرمجة .. فهي لغة صممت أساسا للتفاهم بين علماء الرياضيّات . أما البسكال - والتي بنيت أساسا على لغة الجول ٦٠ - فهي لغة برمجة أعدّت لعمل برمجة مرتبة ومنظّمة . وقد يكون هذا هو السبب الرئيسي في انتشارها السريع ؛ فلم تلق أى لغة من لغات البرمجة مثل هذه السرعة في استخدامها .

ويتوقّع أن يستمر استخدام لغة البسكال استخداما كبيرا مع جميع أجهزة الكمبيوتر خلال الخمسة وعشرين عاما المقبلة بإذن الله تعالى . ونظراً لأهميّة هذه اللغة ، فقد تمت ترجمة هذا الكتاب لتوفير كتاب باللغة العربية في هذه اللغة الهامة للقارئ العربي .

ولايفوتني أن أقدم خالص شكرى لكل من ساهم في هذا العمل لاخراجه فى صورته الحالية ، وأخص بالذكر سعادة الدكتور / عبد الله بن عبد الله العبيد ، عميد كلية الاقتصاد والإدارة لما يوليه من اهتمام كبير لترجمة أمهات الكتب ، وتقديم كل ما يمكن تقديمه من جديد في العلم للقارئ العربي .

وعلى الله قصد السبيل .

المترجم

# مقدمة الكتاب

لقد ظهرت لغة البسكال كالعاصفة في العالم ، منذ وجودها في بداية السبعينيّات . ففى خلال عدّة سنوات بسيطة ، أصبحت اللغة لغة قياسية لطلبة البرمجة الجادين في معظم الكليّات والجامعات . وتدرس هذه اللغة الآن فى العديد من المدارس الثانوية ، وفى الكليّات . كما تَحَوَّل العديد من المشتغلين بالكمبيوتر أيضا إلى البسكال ، مفضّلين هذه اللغة عن لغات أخرى يسهل الوصول إليها . ومن الواضح أنه لم توجد لغة برمجة أخرى أثّرت هذا التأثير الكبير على مجتمع الكمبيوتر فى فترة زمنيّة وجيزة ، مثلما مافعلت لغة البسكال .

ولماذا كل هذا الاهتمام بلغة البسكال . هناك العديد من الأسباب ... أولا لأن لغة البسكال لغة منظّمة . ففى واقع الأمر .. لقد صمّمت لغة البسكال لترويج منهج منظّم ومنسّق لبرمجة الكمبيوتر . ويشجّع استخدامها تطوير البرامج المنظّمة تنظيما جيدًا ، والمكتوبة بوضوح ، والخالية نسبياً من الخطأ سواء استخدمت مع أجهزة الكمبيوتر الكبيرة ، أم الصغيرة ، أم أجهزة الميكروكمبيوتر رخيصة الثمن . ويمكن استخدام اللغة فى أى وسط برمجة بكفاءة مرتفعة ، سواء أكان وسط البرمجة هو تشغيل الدفعة ، أم التشغيل المتداخل . وعلى هذا ... فتقدّم لغة البسكال عددا من الخواص الجيدّة غير الميسّرة نسبياً لبعض لغات البرمجة الأخرى .

ويقدّم هذا الكتاب تعليمات برمجة الكمبيوتر ، مستخدما معالم لغة البسكال القياسيّة (ISO) . والكتاب استمرارًا لكتب البرمجة الأخرى فى مجموعة سلسلة كتب شوم . وعلى هذا .. يقدّم الكتاب توضيحات كاملة ومفهومة لكل المعالم الرئيسيّة للغة البسكال ، مدعّمة بعدد كبير من الأمثلة التوضيحيّة . بالإضافة إلى ذلك ... يقدّم الكتاب منهجا معاصرا للبرمجة ، مع التركيز على أهميّة الوضوح والكفاءة فى إعداد البرنامج . وعلى هذا ... يقدّم للقارئ مبادئ للخبرة فى برمجة جيدّة ، مع تقديم القواعد الخاصة بلغة البسكال .

وقد روعى أن يكون أسلوب الكتابة بسيطا ، حتى يمكن أن يستخدمه الكثير من القرّاء من طلبة المدارس الثانوية ، وحتى المشتغلين بالكمبيوتر ذوى الخبرة . ويناسب الكتاب – بصفة خاصة – طلبة أعلى مستوى فى المدارس الثانوية ، وأول مستوى فى الجامعات ، ككتاب مدرسى مقرّر فى البرمجة ، أو ككتاب مساعد ، أو كدليل فعّال للدراسة الذاتيّة .

وفى معظم أجزاء الكتاب ، لايزيد مستوى الرياضيّات المطلوب عن الجبر الذى يدرس فى المدارس الثانوية . ومعظم المادة المقدّمة لاتتطلّب خلفيّة رياضيّة فى الواقع .

والمادة التى بالكتاب ، مقدّمة بطريقة تجعل القارئ قادرا على كتابة برامج بسكال كاملة وبسيطة بسرعة . ومن المهم جدًا أن يكتب القارئ مثل هذه البرامج وينفذّها أثناء قراءته مادة الكتاب . وسوف يساعد ذلك المبرمجين المبتدئين على اكتساب الثقة بالنفس ، وجذب انتباههم للموضوع ، وذلك بصورة كبيرة . ( ويجب أن يكون مفهوما أن برمجة الكمبيوتر ماهى إلا مهارة ، مثل التأليف القصصى ، أو التأليف الموسيقى . ولايمكن اكتساب مثل هذه المهارة بمجرد قراءة كتاب ) .

ويوجد عدد كبير من الأمثلة كجزء مكمّل للكتاب . وتشمل هذه الأمثلة أمثلة برمجة متعدّدة شاملة ، كما تشمل مشاكل بسيطة تحتاج إلى تفكير . كما أن هناك تمارين برمجة إضافيّة موضحة في المشاكل المحلولة ، وتظهر في نهاية معظم الفصول . هذه الأمثلة والمشاكل المحلولة يجب دراستها بعناية مع استمرار القارئ في قراءة فصول الكتاب ، وبدئه كتابة برامجه الخاصة .

كما يوجد في نهاية كل فصل مجموعة من أسئلة المراجعة ، والمشاكل المتكاملة ، ومشاكل البرمجة . وتمكّن أسئلة المراجعة القارئ من اختبار قدرته على تذكر المادة المقدّمة في كل فصل . كما تقدّم أيضا ملخّصا فعّالا للفصل . ولاتتطلّب معظم المشاكل المتكاملة ومشاكل البرمجة خلفيّة رياضيّة أو تقنية خاصة . ويجب أن يحل الطالب كل ما يستطيع حلّه من هذه المشاكل . ( وتوجد في نهاية الكتاب إجابة للعديد من هذه المشاكل ) . وعند استخدام هذا الكتاب في أحد مقرّرات البرمجة ، قد يرغب الأستاذ في إضافة تمارين برمجة أخرى ، تعكس اهتمامات خاصة بأحد فروع العلم لمشاكل البرمجة .

والمعالم الأساسية للغة ملخّصة في الملاحق من A إلى G ، وموجودة في نهاية الكتاب . ويجب أن يتكرر استخدام هذه المادة كدليل معدّ ، وللتذكرة السريعة . وتكون مفيدة بصفة خاصة عند كتابة أو تصحيح برامج جديدة .

وأخيرا ... فإن القارئ الذي ينتهي من قراءة هذا الكتاب ؛ يصبح لديه فهم متماسك لمفاهيم البرمجة الأساسيّة ، ولقواعد لغة البسكال الخاصة . ومن هذا يستطيع أن يكتسب فهما واقعيًا لإمكانات ومحدّدات الكمبيوتر . كما يجب أن يكون قادرا أيضا على برمجة الكمبيوتر لأداء أنشطة محدّدة ، يقوم باختيارها بنفسه . كما يمكنه أن يمارس بعض الإثارات والإنعاش بنفسه ، بأن يصبح جزءا من ثورة الكمبيوتر الحاليّة ، والتي يبدو أنها ستكون عاملا أساسيًا في إعادة تكوين المجتمعات الصناعيّة الحديثة .

**بايرون س . جوتفريد**

# المحتويات

الفصل الأول

# مفاهيم أولية

# Introductory Concepts

يقدم هذا الكتاب تعليمات فى البرمجة باستخدام لغة برمجة مرتبة ومنظمة ، تسمى بسكال Pascal . وسوف نرى كيف يمكن تحليل وتخطيط ونقل مشكلة موصوفة فى البداية بطريقة غير واضحة إلى برنامج بسكال . وهذه المفاهيم موضحة بالتفصيل عن طريق العديد من عينات المشاكل الموجودة فى الكتاب .

## ١ – مقدمة لأجهزة الكمبيوتر 1.INTRODUCTION TO COMPUTERS

أجهزة الكمبيوتر الحالية متعددة الأشكال والأحجام والتكلفة . فتستخدم أجهزة الكمبيوتر العملاقة متعددة الأغراض فى الشركات الكبيرة والجامعات والمستشفيات والإدارات الحكومية ، وذلك لأداء حسابات علمية ، وحسابات أعمال معقدة . وعادة مايشار إلى هذه الأجهزة على أنها أجهزة كمبيوتر كبيرة mainframes . وهى مرتفعة الثمن ( بعضها يقدر ثمنه بملايين الدولارات ) ، كما أنها تحتاج إلى علميين ومهندسين ومحاسبين معدين لاستخدامها .

وتستخدم أجهزة الكمبيوتر الكبيرة منذ بداية الخمسينيات ، إلا أن قلة من الأفراد أتيحت لهم الفرصة لاستخدامها ، وبصفة خاصة فى السنوات الأولى من استخدامها . وعلى هذا ... فليس من المدهش أن ينظر عامة الناس نظرة غامضة – وبها بعض الارتياب – إلى الكمبيوتر .

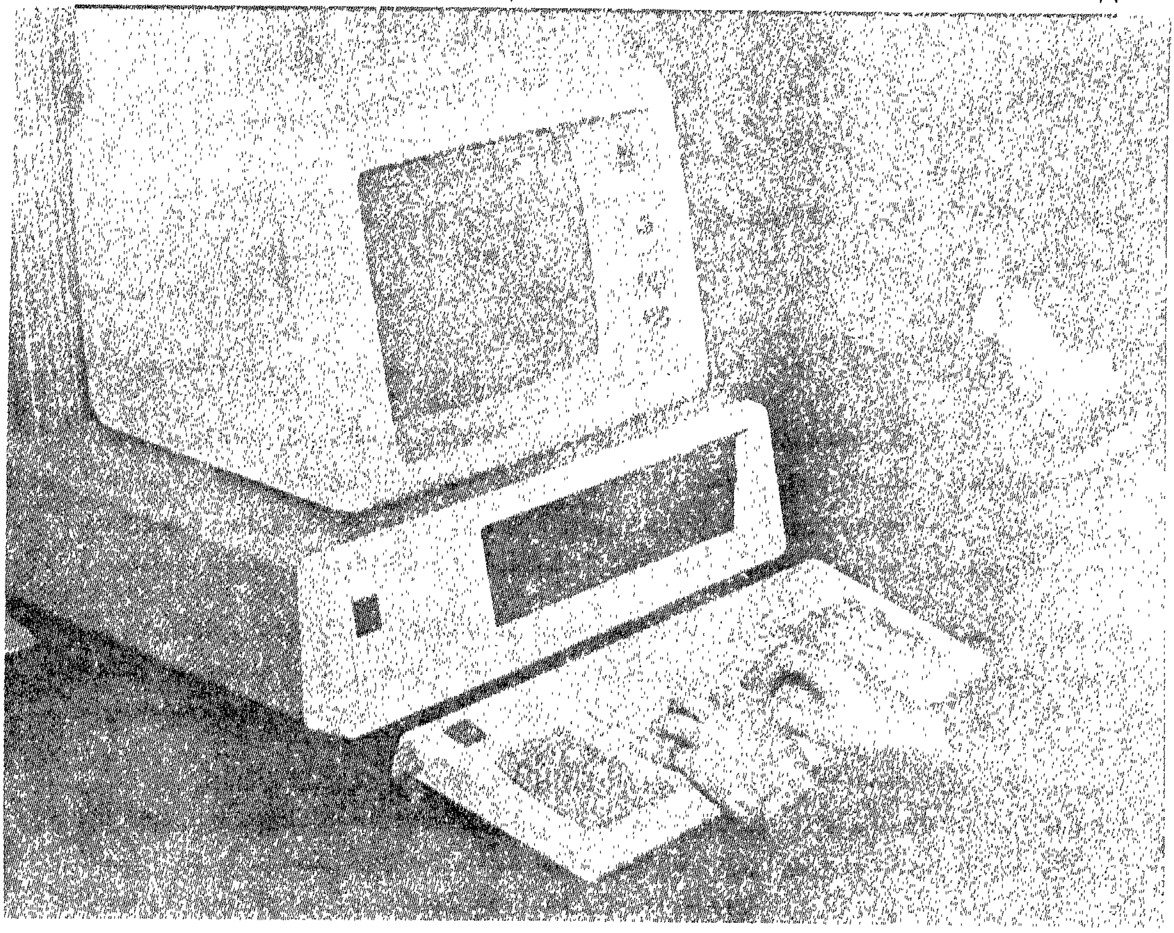
وقد رأت نهاية الستينيات وبداية السبعينيات تطورا لأجهزة مينى كمبيوتر minicomputers ، أصغر حجما وأقل تكلفة . وتقدم الكثير من هذه الآلات نفس أداء أجهزة الكمبيوتر الكبيرة ، وذلك بتكلفة أقل كثيرا من تكلفة هذه الأجهزة . وقد حصل الكثير من الأعمال والمؤسسات التعليمية ، التى ليس لديها المقدرة على الحصول على أجهزة الكمبيوتر الكبيرة ، على أجهزة مينى كمبيوتر ، مع استمرار توفر هذه الأجهزة .

وفى منتصف السبعينيات نتج عن تطور تقنية الدوائر المتكاملة ( رقائق السيليكون ) تطوير لأجهزة كمبيوتر أكثر صغرا ، وأقل تكلفة ، تسمى أجهزة ميكرو كمبيوتر microcomputer . وتتكون هذه الآلات من دوائر متكاملة ، وهى ليست أكبر ( أو أكثر تكلفة ) من آلة كاتبة تقليدية . ويمكن أن تستخدم فى تطبيقات واسعة متعددة ، شخصية ، أو تعليمية ، أو تجارية ، أو تقنية . ويميل استخدامها لاستكمال استخدام أجهزة الكمبيوتر الكبيرة ، وليس للإحلال محلها . وفى واقع الأمر ... فإن العديد من المنظمات الكبيرة تستخدم أجهزة الميكروكمبيوتر كنهايات طرفية أو محطات عمل متصلة بكمبيوتر كبير ( أو بسلسلة من أجهزة الكمبيوتر الكبيرة ) ، وذلك من خلال شبكة اتصالات (*) .

وتطور الكمبيوتر الشخصى Personal Computer يحظى باهتمام خاص ، حيث إن هذه الأجهزة صغيرة ورخيصة ، وتميل إلى أن يستخدمها شخص واحد فى نفس الوقت ( انظر الشكل ١ – ١ ) . وسعه العديد من هذه الآلات تقترب من سعة أجهزة المينى كمبيوتر . وأكثر من هذا ... يستمر الانخفاض فى تكلفتها بصورة كبيرة مع تحسن كبير فى أدائها . وتستخدم أجهزة الكمبيوتر الشخصية حاليا فى الكثير من المدارس ونظم الأعمال الصغيرة . ويبدو أنها سوف تصبح أحد العناصر المنزلية الهامة .

---

(*) لقد دخلت أسواق الكمبيوتر ، مع نهاية عام ١٩٨٩ ، أجهزة ميكروكمبيوتر لها امكانيات تقترب من امكانيات أجهزة الكمبيوتر الكبيرة . ( المترجم ) .



الشكل رقم (١ - ١)

## ٢ - خواص الكمبيوتر 2 - COMPUTER CHARACTERISTICS

كل أجهزة الكمبيوتر عبارة عن وحدات إليكترونية ، يمكنها نقل وتخزين ومعالجة المعلومات ( أى البيانات ) . وهناك نوعان أساسيان مختلفان من البيانات ، وهى : البيانات العددية ، والبيانات الحرفية ( الأسماء ، والعناوين ، وما إلى ذلك ) . وتتطلب التطبيقات العلمية والتقنية تشغيل بيانات عددية أساسا ، بينما تشتمل تطبيقات الأعمال عادة على تشغيل كل من البيانات العددية والحرفية . وتستخدم بعض أجهزة الكمبيوتر فى تشغيل بيانات حرفية فقط ( مثل إعداد الخطابات والكتب وخلافه ) . وهذا مايعرف بتشغيل الكلمات Word Processing

ولتشغيل مجموعة بيانات محددة ، يجب أن يعطى الكمبيوتر مجموعة تعليمات مناسبة تسمى برنامجا Program . ويتم إدخال هذه التعليمات داخل الكمبيوتر لتخزن بعد ذلك فى ذاكرة الكمبيوتر

ويمكن تنفيذ البرنامج المخزن فى أى وقت . وهذا يتسبب فى حدوث الأشياء التالية :

١ - يتم إدخال مجموعة من المعلومات ، تسمى بيانات مدخلات Input Data داخل الكمبيوتر ( عن طريق نهاية طرفية ، أو قارئ بطاقات ، أو غيرها ) . وتخزن فى جزء من أجزاء ذاكرة الكمبيوتر

٢ - بعد ذلك يتم تشغيل بيانات المدخلات لإنتاج نتائج معينة ، تعرف بأنها بيانات المخرجات Output Data

٣ – تابع بيانات المخرجات ( وربما يطبع معها بعض بيانات المدخلات ) على أوراق ، أو تظهر على شاشة موجه monitor ( أي وحدة عرض مرئي ) .

ويمكن إعادة هذه الخطوات الثلاثة العديد من المرات ، إذا كانت هناك رغبة في ذلك لتشغيل كمية بيانات كبيرة في تسلسل سريع . ويجب – على أية حال – فهم أن كل من هذه الخطوات يمكن أن تكون طويلة ومعقدة ، وبصفة خاصة الخطوتان رقم ٢ ، ورقم ٣ .

مثال(١-١)

تم برمجة كمبيوتر لحساب مساحة الدائرة باستخدام العلاقة : $A = \pi r^2$ ، وذلك إذا ماأعطيت القيمة العددية لنصف القطر كبيانات مدخلات . يتطلب ذلك اتباع الخطوات التالية :

١ – قراءة قيمة عددية لنصف قطر الدائرة .

٢ – حساب قيمة المساحة ، باستخدام العلاقة سابقة الذكر ( سوف تخزن هذه القيمة مع بيانات المدخلات في ذاكرة الكمبيوتر ) .

٣ – طباعة ( أو عرض ) قيمة نصف القطر ، وقيمة المساحة المناظرة له .

٤ – التوقف .

تتطلب كل خطوة من هذه الخطوات إحدى تعليمات – أو العديد من تعليمات – برنامج الكمبيوتر .

توضح المناقشة السابقة سمتين هامتين الكمبيوتر ، هما : الذاكرة ، وإمكانية البرمجة . كما أن السرعة والاعتمادية من السمات الهامة الأخرى . وسوف نذكر المزيد عن الذاكرة والسرعة والاعتمادية في المقاطع القليلة التالية . أما موضوع إمكانية البرمجة ، فسوف يناقش بالتفصيل خلال ماتبقى من هذا الكتاب .

## الذاكـرة : Memory

كل المعلومات المخزنة داخل ذاكرة الكمبيوتر ، تخزن على هيئة شفرة من خليط من الأصفار والرقم 1 . وتسمى هذه الشفرة ( أصفار وأرقام 1 ) بالبت ، وهي اختصار للأرقام الثنائية binary digits ، وتمثل كل بت تمثيلا إليكترونيا بحيث إنها تكون في الوضع الموصل On الذي يمثل الرقم 1 . أو وضع الفصل Off الذي يمثل الصفر .

ومعظم أجهزة الكمبيوتر الصغيرة لها ذاكرات مرتبة طبقا لتكرار كل ثمانية بت . وتسمى كل ثمانية بت بالبايت byte . وعادة مايمثل البايت الواحد رمزا واحدا ( أي يمثل حرفا ، أو رقما ، أو رمزا خاصا ) . ويمكن أن تحتل إحدى التعليمات 1 أو 2 أو 3 بايت ، كما أن الكميات العددية يمكنها أن تحتل من 2 إلى 8 بايت ، وذلك طبقا لدقة العدد ونوعه .

وعادة مايعبر عن حجم سعة ذاكرة الكمبيوتر بمضاعفة $2^{10}$ ، وهو 1024 بات . ويشار إلى هذا العدد بالرمز 1 KB . وعادة ماتتراوح سعة ذاكرة أجهزة الكمبيوتر الصغيرة من 64 KB إلى 1024 KB ، أي إلى 1 MB (*) .

مثال (١-٢)

سعة جهاز كمبيوتر شخصي صغير هي 64 KB . وعلى هذا ... فيمكن تخزين 64 x 1024 = 65,536 رمزا أو عددا من التعليمات في ذاكرة هذا الكمبيوتر . فاذا مااستخدمت الذاكرة لتمثيل بيانات حرفية ، فيمكن تخزين حوالي 800 اسم وعنوان داخل ذاكرة الكمبيوتر في آن واحد ( وذلك بفرض أن كل اسم وعنوان يشغلان 80 خانة ) .

(*) ازدادت هذه السعة مع نهاية عام ١٩٨٩ ليمكن أن تصل الى اكثر من 200 MB ( المترجم ) .

أما أجهزة الكمبيوتر الكبيرة ، فلديها ذاكرات مختلفة على هيئة كلمات Words ، وليس على هيئة بايت ، يحتوى كل كلمة على عدد كبير نسبيا من البت ، وعادة مايكون 32 أو 36 . وهذا يسمح بتمثيل مجموعة بسيطة من الحروف ( عادة أربعة أو خمسة ) داخل ذاكرة مكونة من كلمة واحدة . وعادة مايعبر عن سعة أجهزة الكمبيوتر الكبيرة بمضاعف ($2^{10}$ = 1024 k words) ، فجهاز الكمبيوتر الكبير يمكن أن تكون سعة ذاكرته العديد من ملايين من الكلمات .

مثال (١-٢)

سعة ذاكرة جهاز كمبيوتر ذى أغراض عامة ، هى 2048 KW ، وهى تناظر 2,097,152 = 2048 x 1024 كلمة . فإذا ما استخدمت الذاكرة لتمثيل بيانات عددية ، فيمكن تخزين حوالى ٢ مليون عدد داخل الكمبيوتر فى أى وقت .

أما إذا استخدمت الذاكرة لتمثيل بيانات حرفية ، بدلا من البيانات العددية ، فيمكن تخزين حوالى ٨ مليون حرف فى نفس الوقت . وهذه الذاكرة تكفى لتخزين محتويات كتاب كامل ، بل تزيد عن ذلك أيضا .

كما تستخدم معظم أجهزة الكمبيوتر وحدات تخزين مساعدة auxiliary Storage Devices أيضا ( مثل الشرائط المغناطيسية ، والأقراص المغناطيسية ، وغيرها ) ، وذلك بالإضافة إلى ذاكرتها الابتدائية . وعادة ماتتراوح سعة هذه الوحدات من عدة مئات من الآلاف من البايت ( بالنسبة لأجهزة الكمبيوتر الصغيرة ) إلى عدة ملايين من الكلمات ( بالنسبة لأجهزة الكمبيوتر الكبيرة ) . وأكثر من هذا ... فإنها تسمح بتسجيل المعلومات تسجيلا دائما ، حيث يمكن تركيبها على الكمبيوتر ، أو فكها منه وتخزينها لحين الحاجة إليها . وعلى أية حال ... فإن زمن الاتصال ( وهو الزمن اللازم لتخزين المعلومات أو لاسترجاعها ) يكون كبيرا لهذه الوحدات الثانوية ، بالمقارنة لقرينه الخاص بالذاكرة الابتدائية .

## السرعة والاعتمادية : Speed and Reliability

نظرا لسرعة الكمبيوتر المرتفعة جدا ، فإنه يستطيع أداء الحسابات التى تحتاج إلى أشهر لأدائها يدويا فى عدة دقائق . والأنشطة البسيطة ، مثل جمع عددين ، يمكن أن تؤدى فى جزء من الميكروثانية ( 1 ميكروثانية = $10^{-6}$ ثانية ) . وعلى معظم المستويات العملية ، فإن درجات الطلبة فى نهاية الفصل الدراسى فى إحدى الجامعات الكبيرة ، يمكن إعدادها فى عدة دقائق بالنسبة للكمبيوتر .

وهذه السرعة العالية جدا يصاحبها درجة اعتمادية على نفس المستوى . فالكمبيوتر لايخطئ عمليا على الإطلاق . وأخطاء الكمبيوتر التى يركز عليها فى وسائل الإعلام ، مثل تسلم أحد الأفراد فاتورة تفيد بأن مديونيته وصلت مليون دولار لأحد محلات البيع بالتجزئة ، ماهى إلا نتيجة لخطأ فى البرمجة أو فى النقل ، وليست بسبب خطأ الكمبيوتر نفسه .

## ٣ – طرق التشغيل 3 MODES OF OPERATION

هناك طريقتان مختلفتان يمكن أن تستخدم بهما وسائل الكمبيوتر . هاتان الطريقتان هما ، طريقة الدفعة وطريقة التداخل . وكل من الطريقتين شائعة الاستخدام . كما أن كل منهما لها مميزاتها بالنسبة لنوع معين من المشاكل .

### تشغيل الدفعة Batch Processing

فى الأيام الأولى لاستخدام الكمبيوتر ، كانت كل الأعمال تنفذ عن طريق تشغيل الدفعة batch processing ومازالت هذه الطريقة مستخدمة حتى الآن ، بالرغم من قلة استخدامها بالنسبة لما كانت عليه فى الأيام الأولى لاستخدام الكمبيوتر .

وفي تشغيل الدفعة ، يقرأ العديد من الأعمال داخل الكمبيوتر ، وتخزن هذه الأعمال داخليا ، ثم يتم تشغيلها على التوالى . ( وتشير الأعمال إلى برامج كمبيوتر ، وما يصاحبها من مجموعات من بيانات المدخلات التى يجرى عليها التشغيل ) . ويتطلب تشغيل الدفعة أن تكون البرامج والبيانات مسجلة على بطاقات مثقبة . وتقرأ هذه المعلومات داخل الكمبيوتر بواسطة قارئ البطاقات ، ثم بعد ذلك يتم تشغيلها . وبعد الانتهاء من تشغيل أحد الأعمال ؛ تطبع المخرجات ومعها قائمة بالبرنامج على شريط طويل من الأوراق بواسطة طابع ذى سرعة طباعة عالية .

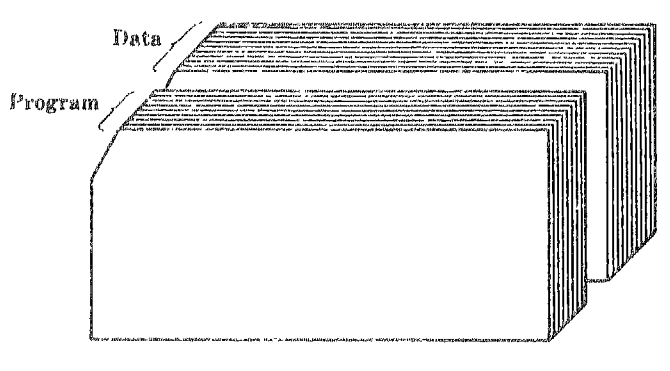
أما تشغيل الدفعة الحديث ، فيرتبط بصفة عامة بنظام المشاركة الزمنية . وفى هذا النظام يتم إدخال البرنامج ومجموعة البيانات داخل الكمبيوتر عن طريق نهاية طرفية أو ميكروكمبيوتر . وبعد ذلك تخزن المعلومات داخل ذاكرة الكمبيوتر ، وتنفذ طبقا لترتيبها المناسب . وتفضل هذه الصيغة من صيغ تشغيل الدفعة عن الطريقة التقليدية ، حيث إنها تلغى الحاجة إلى البطاقات المثقبة ، وتسمح بتنقيح معلومات المدخلات ( البرنامج والبيانات ) أثناء إدخالها .

ويمكن نقل كميات كبيرة من المعلومات ( البرامج والبيانات ) داخل الكمبيوتر ، ومنه إلى الخارج بسرعة عالية فى تشغيل الدفعة . والأكثر من هذا هو أن المستفيد لايكون فى حاجة لأن يتواجد أثناء تنفيذ العمل الخاص به . وعلى هذا .. فإن مثل هذه الطريقة تناسب الأعمال التى تتطلب وقتا طويلا من وقت الكمبيوتر ، أو أنها تكون طويلة ومن ناحية أخرى ... قد يختلف إجمالى الوقت اللازم لتشغيل أحد الأعمال بهذه الطريقة من عدة دقائق إلى العديد من الساعات ، وذلك بالرغم من أن هذا العمل قد لايحتاج إلا إلى ثانية واحدة أو اثنتين من الوقت الفعلى لتشغيل الكمبيوتر ( يجب أن ينتظر كل عمل دوره قبل أن يمكن قراءته وتشغيله وطباعة مخرجاته ) . وعلى هذا ... فقد لايكون مرغوبا فيه تشغيل الدفعة عندما يكون هناك حاجة إلى تشغيل العديد من الأعمال الصغيرة البسيطة ، والحصول على النتائج بأسرع مايكون .

مثال (١–٤)

لدى أحد الطلاب ١٠٠٠ قيمة مختلفة لنصف قطر دائرة ، ويريد أن يحسب مساحة الدائرة لكل قيمة من هذه القيم . لعمل ذلك ... يجب عليه أن ينفذ برنامج كمبيوتر يشبه البرنامج الذى سبق وصفه فى المثال (١ – ١) . وتجرى الحسابات فى هذه الحالة ١٠٠٠ مرة ، بمعدل مرة واحدة لكل قيمة من قيم نصف القطر ( ويجب أن يعدل البرنامج ليمكن تكرار تنفيذه ) . يستخدم تشغيل الدفعة ، حيث إن كل الحسابات سوف تجرى مرة واحدة . دعنا نفترض أن البرنامج والبيانات على بطاقات مثقبة .

لتشغيل البيانات ، سوف يقرأ الطالب مجموعة البطاقات داخل الكمبيوتر . وأول جزء من مجموعة البطاقات يحتوى على البرنامج . ويلى البرنامج ١٠٠٠ بطاقة بيانات ، كل منها يحتوى على قيمة واحدة لنصف القطر . ومجموعة البطاقات هذه موضحة فى الشكل رقم (١–٢) .




الشكل رقم (١–٢)

وبعد قراءة مجموعة البطاقات داخل الكمبيوتر ، قد يحدث تأخير لمدة ساعات قبل أن ينفذ البرنامج ، ويتم تشغيل البيانات . وبعد الانتهاء من إجراء الحسابات ، تطبع النتائج على شريط كبير من الأوراق . وفي هذه الحالة يتلقى الطالب قائمة تحتوى على البرنامج الفعلى ، ويتبعه ١٠٠٠ زوجا من الأعداد . كل زوج من هذه الأزواج من الأعداد يمثل قيمة واحدة لنصف القطر ، ومساحة الدائرة المناظرة لها . يبين الشكل رقم (١-٣) جزءا من مخرجات الكمبيوتر .

| RADIUS | AREA |
|---|---|
| 1 | 3.141593 |
| 2 | 12.566372 |
| 3 | 28.274337 |
| 4 | 50.265488 |
| 5 | 78.5398251 |
| 6 | 113.097348 |
| 7 | 153.938057 |
| 8 | 201.061952 |
| 9 | 254.469033 |
| 10 | 314.1593 |
| 11 | 380.132753 |
| 12 | 452.389392 |
| 13 | 530.929217 |
| 14 | 615.752229 |
| 15 | 706.858425 |
| 16 | 804.247808 |
| 17 | 907.920378 |
| 18 | 1017.87613 |
| 19 | 1134.11507 |
| 20 | 1256.6372 |

الشكل رقم (١-٣)

## التشغيل المتداخل Interactive Computing

يحدث التشغيل المتداخل Interactive Computing باستخدام كمبيوتر شخصى ، مثل الموضح فى الشكل رقم (١ - ١) ، أو باستخدام نهاية طرفية لكمبيوتر كبير ، كما فى الشكل رقم (١-٢) . وفى أى من الحالتين ، يقدم المستفيد معلومات المدخلات للكمبيوتر من خلال لوحة المفاتيح التى تشبه مفاتيح الآلة الكاتبة . ومعلومات المخرجات المناظرة يتم طباعتها على شريط طويل من الأوراق ، أو تعرض على موجه مرئى ( قد يكون من المرغوب فيه الحصول على مخرجات مطبوعة لبعض التطبيقات ، حيث إن ذلك يقدم نسخة دائمة من التشغيل المتداخل . وعلى أية حال . . فإن استخدام الموجه عادة مايكون أكثر راحة ) . وعادة مايشار إلى النهايات الطرفية المستخدمة فى التشغيل المتداخل ، بأنها نهايات طرفية مرئية Console .

إحدى السمات المعنوية للتشغيل المتداخل هى أن المستفيد يكون قادرا على إجراء حوار مع الكمبيوتر أثناء التشغيل . وعلى هذا . . . فيمكن للمستفيد أن يسأل الكمبيوتر بصفة دورية أن يقدم له معلومات معينة تحدد الإجراء التالى الذى سيتخذه الكمبيوتر



الشكل رقم (١-٤)

مثال (١-٥)

يرغب أحد الطلاب في استخدام الكمبيوتر في حساب نصف قطر الدائرة التي مساحتها ١٠٠ . والبرنامج الذي يحسب مساحة الدائرة إذا ماأعطى نصف القطر موجود . ( لاحظ أن الموجود هو عكس مايريده الطالب تماما ) . وهذا البرنامج ليس هو البرنامج المطلوب ، إلا أنه يسمح للطالب أن يعمل عن طريق التجربة والخطأ ، حتى يجد قيمة نصف قطر الدائرة التي تقترب مساحتها من ١٠٠ .

وبمجرد إدخال البرنامج المطلوب تظهر الرسالة التالية .

RADIUS=?

عند ذلك يقوم الطالب بإدخال قيمة نصف القطر . دعنا نفترض أن الطالب قام بإدخال قيمة ٥ لنصف القطر . عند ذلك يستجيب الكمبيوتر الطباعة .

```
AREA= 78.5398

DO YOU WISH TO REPEAT THE CALCULATION?
```

عند ذلك يكتب الطالب "yes" أو "no" ، فإذا ماكتب الطالب "yes"؛ تظهر له الرسالة التالية :

```
RADIUS=?
```

وتعاد نفس الإجراءات . أما إذا ماكتب الطالب "no" ، فتظهر له الرسالة التالية :

```
GOODBYE
```

وينفصل البرنامج عن الكمبيوتر .

ويوضح الشكل رقم (١-٥) المعلومات التى تطبع أثناء عملية متداخلة تقليدية باستخدام البرنامج سالف الذكر ، مع وضع خط تحت الكلمات التى كتبها الطالب . وقد تحددت قيمة تقريبية لنصف القطر تساوى ٥,٦ بعد ثلاث محـاولات .

INTRODUCTORY CONCEPTS

```
RADIUS=? 5
AREA= 78.5398

DO YOU WISH TO REPEAT THE CALCULATION? YES

RADIUS=? 6
AREA= 113.097

DO YOU WISH TO REPEAT THE CALCULATION? YES

RADIUS=? 5.6
AREA= 98.5204

DO YOU WISH TO REPEAT THE CALCULATION? NO

GOODBYE
```

الشكل رقم (١-٥)

لاحظ الطريقة التى تبدو كحوار بين الطالب والكمبيوتر . ولاحظ أيضًا أن الطالب ينتظر حتى يرى قيمة المساحة المحسوبة ، قبل أن يقرر ماإذا كانت ستجرى حسابات أخرى أم لا . فإذا مابدأت عملية حسابات أخرى ، فتعتمد القيمة الجديدة لنصف القطر الذى يقدمها الطالب على النتائج التى سبق الحصول عليها .

وفى بعض الأحيان ، يقال عن البرامج التى تصمم لتطبيقات التشغيل المتداخل بأنها ذات طبيعة تحاور Conversational . والمباريات التى تجرى مع الكمبيوتر – مثل لعب الشطرنج – هى أمثلة ممتازة لمثل تطبيقات التداخل هـذه .

## المشاركة الزمنية Timesharing

المشاركة الزمنية timesharing هى صيغة من صيغ التشغيل المتداخل ، والتى يكون العديد من المستفيدين قادرين فيها على استخدام كمبيوتر واحد فى نفس الوقت . ويتصل كل مستفيد بالكمبيوتر من خلال نهاية طرفية ، مثل النهاية الموضحة فى الشكل رقم (١- ٤) ، أو من خلال ميكرو كمبيوتر ، مثل الموضح فى الشكل رقم (١ - ١) . ويمكن أن تكون النهايات الطرفية متصلة سلكيا بالكمبيوتر مباشرة ، أو يمكن أن تكون متصلة بالكمبيوتر عبر خطوط الهاتف ، أو عبر دوائر ميكروويف . وعلى هذا ... فيمكن للنهاية الطرفية أن توضع بعيدا عن الكمبيوتر المستخدم . وربما تصل هذه المسافة إلى العديد من مئات الأميال .

وحيث إن الكمبيوتر يعمل بطريقة أسرع من عمل النهاية الطرفية ، فيمكن لجهاز كمبيوتر واحد أن يخدم عدداً كبيراً من النهايات الطرفية فى نفس الوقت . وعلى هذا ... لايكون أى مستفيد قلقا من ناحية تواجد مستفيدين آخرين . وسوف يبدو له أن الكمبيوتر كما لو كان يعمل له هو بمفرده .

وتناسب المشاركة الزمنية تشغيل الأعمال الصغيرة نسبيا ، والتى لاتتطلب نقلا كبيرا للبيانات ، أو تتطلب وقتا كبيرا من الكمبيوتر . ومعظم تطبيقات الكمبيوتر التى تظهر فى المدارس والجامعات والمكاتب التجارية تتميز بمثل هذه السمات . ويمكن تشغيل مثل هذه التطبيقات بسرعة وبسهولة وبتكلفة أقل إذا مااستخدم تشغيل المشاركة الزمنية .

مثال (١-٦)

لدى إحدى الجامعات الكبيرة إمكانية المشاركة الزمنية فى كمبيوتر محتوٍ على ١٠٠ نهاية طرفية للمشاركة الزمنية . وتوجد هذه النهايات الطرفية فى المواقع المختلفة للجامعة . وتتصل النهايات الطرفية بجهاز كمبيوتر عبر خطوط الهاتف . وتنقل كل نهاية طرفية بيانات إلى الكمبيوتر أو منه ، بسرعة قصوى تعادل ١٢٠ رمزا فى الثانية . ويمكن استخدام جميع النهايات الطرفية فى نفس الوقت ، بالرغم من أنها متداخلة مع كمبيوتر واحد .

بالإضافة إلى ذلك ... تتصل بالكمبيوتر ٢٠ نهاية طرفية ، تقع فى أماكن بعيدة ، حيث يوجد خمسة عشر من هذه النهايات الطرفية فى خمسة مواقع فى منطقة تواجد الكمبيوتر . أما الخمس نهايات الطرفية الأخرى ، فتقع فى معامل أبحاث حكومية . ويمكن استخدام كل النهايات الطرفية – والبالغ عددها ١٢٠ – فى نفس الوقت ( وعادة مايحدث ذلك ) . وبالمشاركة بهذه الطريقة فى الكمبيوتر ، يكون كل مستفيد قادرا على استخدام خدمة الكمبيوتر الكبير بتكلفة معقولة .

ويمكن دمج بعض معالم تشغيل الدفعة والمشاركة الزمنية إذا ماكانت هناك رغبة فى ذلك . فيمكن على سبيل المثال ... إدخال مجموعة من بيانات المدخلات من نهاية طرفية مباشرة ، على أن يتم تشغيلها بطريقة الدفعة الذى سبق وصفها . وهناك إمكانية أخرى ، وهى استخدام قارئ بطاقات ( تشغيل الدفعة ) فى إدخال برنامج ومجموعة بيانات ، ثم تعديل البرنامج ، وتشغيل البيانات بطريقة المشاركة الزمنية . مثل هذه العمليات أصبحت شائعة الاستخدام فى الوقت الحديث .

## ٤ – أنواع لغات البرمجة 4- TYPES OF PROGRAMMING LANGUAGES

هناك العديد من اللغات المختلفة التى يمكن استخدامها فى برمجة الكمبيوتر . وأكثر هذه اللغات قربا للكمبيوتر هى لغة الآلة machine language ، وهى مجموعة من التعليمات التفصيلية جدا الخفية ، التى تتحكم فى دوائر الكمبيوتر الداخلية . وهذه هى اللهجة الطبيعية للكمبيوتر . ويكتب القليل جدا من برامج الكمبيوتر فى واقع الأمر بهذه اللغة لسببين : أولا : أن لغة الآلة مرهقة جدا فى العمل بها . وثانيا : أن معظم أجهزة الكمبيوتر لديها تعليماتها الخاصة بها . وعلى هذا فإن البرنامج المكتوب بلغة الآلة لأحد أنواع أجهزة الكمبيوتر لايمكن تشغيله على نوع آخر من أنواع أجهزة الكمبيوتر ، دون إدخال تعديلات جوهرية عليه .

وعادة مايكتب برنامج الكمبيوتر بإحدى اللغات مرتفعة المستوى high - level language ، والتى تكون تعليماتها أكثر توافقية مع اللغات الآدمية . ومعظم هذه اللغات مرتفعة المستوى هى لغات ذات أغراض عامة general - purpose ، مثل لغة البسكال . ( وهناك لغات ذات أغراض عامة أخرى ، مثل : البيسك ، والكوبل ، والفورتران ، ولغة PL/1 ) . كما يوجد لغات أخرى مرتفعة المستوى ذات أغراض خاصة special - purpose ، والتى تصمم تعليماتها لتناسب أحد أنواع التطبيقات المحددة بصفة خاصة .

وكقاعدة عامة ... فإن إحدى تعليمات أى لغة ذات مستوى مرتفع ، تعادل عدة تعليمات بلغة الآلة . والأكثر من هذا ... فيمكن تشغيل البرنامج المكتوب بإحدى اللغات مرتفعة المستوى على العديد من أجهزة الكمبيوتر ، وذلك مع إدخال بعض التعديلات البسيطة ، أو بدون إدخال أى تعديلات بالمرة . وعلى هذا ... فإن استخدام اللغات مرتفعة المستوى يقدم بعض المميزات المعنوية جدا لنا عن استخدم لغة الآلة ، وبالتحديد فإنه يقدم لنا البساطة والانتظام ، وإمكانية استخدام البرنامج مع العديد من أجهزة الكمبيوتر ( أى عدم اعتماد البرنامج على الكمبيوتر نفسه ) .

ويجب - على أية حال - أن يترجم البرنامج المكتوب بلغة مرتفعة المستوى إلى لغة الآلة قبل أن يكون قابلا للتنفيذ . وهذا مايعرف بالترجمة Compilation ، أو بالتفسير Interpretation طبقا لكيفية أداء عملية الترجمة . ( معظم صيغ البسكال يتم لها عملية ترجمة ، وليست عملية تفسير ) .

وبصفة عامة ... فإنه من المريح أكثر إعداد البرنامج الجديد باستخدام مفسر ، بدلا من استخدام مترجم ، وذلك بالرغم من أن البرنامج المترجم ينفذ بصورة أسرع كثيرا من البرنامج المفسر . ( أسباب ذلك لاتقع فى مدى مناقشتنا الحالية ) . وفى أى حالة من الحالتين - على أية حال - تتم عملية الترجمة تلقائيا داخل الكمبيوتر . وفى واقع الأمر لايكون المبرمج المبتدئ مهتما بمثل هذه العملية على الإطلاق ، حيث إنه يرى البرنامج الأصلى فقط ، ويرى بيانات المدخلات ، وبيانات المخرجات التى يحصل عليها من البرنامج فقط .

والمترجم أو المفسر فى حد ذاته ماهو إلا برنامج كمبيوتر يقبل برنامج كمبيوتر مكتوبا بلغة مرتفعة المستوى كبيانات مدخلات ، وينتج عنه نفس البرنامج بلغة الآلة كمخرجات . وطبقا لذلك ... يسمى البرنامج الأصلى مرتفع المستوى ببرنامج المصدر Source ، ويسمى البرنامج الناتج بلغة الآلة ببرنامج التشغيل Object . ويجب أن يكون لكل جهاز كمبيوتر مترجمه أو مفسره الخاص بالنسبة للغة مرتفعة المستوى المستخدمة معه . وهذا الاستخدام للمترجمات أو المفسرات هو الذى يسمح لنا بتحقيق الانتظام وعدم الاعتماد على الكمبيوتر بالنسبة للغات مرتفعة المستوى ، مثل البسكال .

## ٥ - مقدمة للغة البسكال 5 - INTRODUCTION TO PASCAL

لغة البسكال هى لغة تستخدم فى الأغراض العامة ، كما أنها لغة مرتفعة المستوى . وقد استخلصت هذه اللغة من لغة Algol-60 . وتعد تعليمات لغة البسكال بطريقة تشبه التعبيرات الجبرية ، مستخدمة بعض كلمات اللغة الإنجـليزية ، مثل الكلمات التالية :

BEGIN, END, read, write, IF, THEN, REPEAT, WHILE, DO.

وبهذا الشكل فإن لغة البسكال تشبه العديد من اللغات ذات المستوى المرتفع . كما تحتوى لغة البسكال على بعض المعالم الخاصة بها ، والتى صممت خصيصا لتشجيع استخدام البرمجة المرتبة Structured Programming . والبرمجة المرتبة هى منهج مرتب ومنظم لإعداد برامج واضحة ، ومرتفعة الكفاءة ، وخالية من الأخطاء . ولهذا السبب يفضل العديد من المعلمين والمبرمجين المهنيين استخدام لغة البسكال عن لغات الأغراض العامة الأخرى .

وقد سميت لغة البسكال باسم العالم الفرنسى المشهور ( بليز بسكال ) الذى ولد فى عام ١٦٢٣م ، وتوفى فى عام ١٦٦٢م ، والذى حقق العديد من الاكتشافات العلمية ، والتى منها اختراع أول آلة حاسبة ميكانيكية فى العالم .

## نشأة لغة البسكال وتطورها History of Pascal

أعد لغة البسكال فى بدايتها ( نيكلوس ويرث Niklous Wirth ) فى جامعة التقنية فى زيورخ فى بداية السبعينيات . وقد كان الهدف الأساسى لويرث هو تطوير لغة منظمة مرتفعة المستوى لتعليم البرمجة المرتبة . وأحيانا يشار إلى التعريف الأساسى للغة الذى أعده ويرث بأنه البسكال القياسية Standard Pascal ، أو البسكال القياسية كما يعرفها ويرث وجنسين (٣) Standard Pascal as defined by Jensen and Wirth . وعلى أية حال ... فهناك بعض الغموض فى اصطلاح بسكال القياسية ، حيث إنه يوجد العديد من القياسات المختلفة فى وقتنا الحالى .

وتختلف معظم مترجمات البسكال بعض الشئ عن التعريف الأساسى لويرث . فقد اقترحت المؤسسة العالمية للقياسات International Standards Organization (ISO/DIS7185) صيغة قياسية أوربية حديثة . وفى فترة كتابة هذا الكتاب ، كانت هناك صيغة قياسية أمريكية تحت الإعداد بواسطة المعهد القومى الأمريكى للقياسات American National Standards Institute (ANSI) بالمشاركة مع معهد الهندسة الكهربائية والإلكترونية Institute of Electrical and Electronic Engineers (IEEE) . ويبدو أن الصيغة القياسية الأمريكية سوف تختلف اختلافا بسيطا عن الصيغة القياسية الأوربية .

وحاليا تستخدم لغة البسكال بصورة واسعة فى الولايات المتحدة الأمريكية وفى أوربا ، وذلك كلغة تعليمية ولغة قوية للاستخدام فى الأغراض العامة فى العديد من التطبيقات المختلفة . وقد ازدادت شعبية استخدامها مع كل من أجهزة الكمبيوتر الكبيرة والصغيرة . وفى واقع الأمر ... فإن لغة البسكال أصبحت شائعة الاستخدام من قبل المستفيدين من أجهزة الميكروكمبيوتر وهناك بعض التوقعات بأنها سوف تحل محل لغة البيسك كلغة سائدة مع أجهزة الميكروكمبيوتر خلال السنوات القليلة التالية .

وهذا الكتاب موجه بصفة مبدئية نحو استخدام صيغتى البسكال القياسيتين ISO/ANSI . وحيث إن هاتين الصيغتين القياسيتين ليستا شائعتا الاستخدام ، فسوف نناقش السمات غير القياسية أيضا . وتقدم هذه المادة الأساس لكل المترجمات التجارية للغة البسكال . وعلى هذا ... فإن القارئ الذى يلم بهذه المادة لن يكون أمامه سوى صعوبة بسيطة ليتعلم أى صيغة أخرى من صيغ هذه اللغة .

## ترتيب برنامج البسكال Structure of a Pascal Program

يحتوى كل برنامج مكتوب بلغة البسكال على عنوان ومجموعة . ويبدأ العنوان header بكلمة PROGRAM ، ويتبعها بعض المعلومات الإضافية المطلوبة . ويكتب هذا الجزء من البرنامج فى سطر واحد فقط .

أما المجموعة block ، فتحتوى على جزئين أساسيين ، وهما : جزء التوضيح ، وجزء العبارات . ويعرف جزء التوضيح Declaration part عناصر البيانات المختلفة المستخدمة فى البرنامج . أما جزء العبارات Statement part فيحتوى على العبارات الفعلية التى تتسبب فى اتخاذ إجراءات . ويجب أن تظهر جملة واحدة على الأقل فى كل برنامج مكتوب بلغة البسكال . والترتيب الشامل للبرنامج موضح بتفصيل أكثر فى التخطيط التالى :

1. Header
2. Block
   (*a*) Declarations
      - Labels
      - Constants
      - Type definitions
      - Variables
      - Procedures and functions

   (*b*) Statements

---

(٣) Jensen, K. and N.Wirth "Pascal User Manual and Report", 2nd. edition, Springer - Verlag, 1974.

(٤) مامن احد ينكر الانتشار السريع للغة البسكال ، الا انه من المتوقع ان يزداد الاقبال على كل من لغة البسكال ولغة البيسك خاصة بعد ادخال الكثير من التعديلات على صيغها مثل بيسك السريع وتربوبيسك ( المترجم ) .

وسوف نناقش كل عنصر من هذه العناصر للبرنامج بتفاصيل أكثر فيما بعد فى هذا الكتاب . وكل مايهمنا الآن هو الشكل العام فقط . وتجب الإشارة إلى أنه ليست هناك حاجة لظهور مكونات جزء التوضيح كلها فى كل برنامج من برامج البسكال . فإذا وجدت على أية حال ، فيجب أن تظهر بنفس الترتيب الموجود أعلاه .

## مثال (١-٧)

مساحة الدائرة . يوجد هنا برنامج أولى بلغة البسكال يسمى Circle ، ويقرأ نصف قطر الدائرة ، ويحسب مساحتها ، ثم يكتب قيمة كل من نصف القطر والمساحة .

```
PROGRAM circle(input,output);          (* HEADER *)
VAR area,radius : real;                (* VARIABLE DECLARATION *)
BEGIN
   read(radius);                       (* STATEMENT *)
   area := 3.14159*sqr(radius);        (* STATEMENT *)
   write(radius,area)                  (* STATEMENT *)
END.
```

المكونات الأساسية لهذا البرنامج تم فصلها وتسميتها بالترتيب لتمييز التنظيم الشامل للبرنامج . وفى العادة لايأخذ البرنامج هذا الشكل . وبدلا من ذلك ، فقد يظهر البرنامج على النحو التالى :

```
PROGRAM circle(input,output);
(* PROGRAM TO CALCULATE THE AREA OF A CIRCLE *)
VAR area,radius : real;
BEGIN
   read(radius);
   area := 3.14159*sqr(radius);
   write(radius,area)
END.
```

وتجب الإشارة إلى السمات التالية بالنسبة لهذا البرنامج :

١ – بعض الكلمات مكتوبة بحروف كبيرة . وهذه هى الكلمات المحجوزة reserved words أو الكلمات الأساسية Keywords ، والتى لها معنى سبق تحديده ( وسوف تذكر بالتفصيل فيما بعد ) .

٢ – يحتوى السطر الأول على اسم البرنامج (circle) ، وعلى بعض المعلومات الإضافية التى سوف يتم وصفها فى القسم التالى . وهذا هو عنوان البرنامج .

٣ – السطر الثانى هو تعليق Comment يعرف الغرض من البرنامج . ويمكن تمييز التعليقات بصفة دائمة ، حيث إنها توضع بين رمزين خاصين هما : ( * ... * ) .

٤ – لاحظ الثلاثة أسطر المرحلة ، والموجودة بين END,BEGIN . هذه هى عبارات Statements البرنامج . فهى تتسبب فى إدخال قيمة نصف القطر فى الكمبيوتر ، وفى حساب قيمة المساحة ، وفى كتابة قيمة نصف القطر وقيمة المساحة . وترحيل هذه العبارات غير ضرورى ، إلا أنه يوصى بشدة بعمله كطريقة جيدة للبرمجة العملية . ( وهذا هو نوع تطوير البرنامج المنظم ، والتى صممت لغة البسكال لتشجع على استخدامه ) .

٥ – تمثل القيم العددية لنصف القطر والمساحة بالأسماء الرمزية area , radius . وتسمى هذه الأسماء الرمزية بالمتغيرات Variables ، وهى معرفة أو موضحة فى السطر الثالث من البرنامج .

٦ - الاسم الرمزى sqr الموجود فى السطر السادس هو دالة قياسية Standard Function تستخدم لحساب مربع نصف القطر .

٧ - لاحظ فى النهاية التنقيط الموجود فى نهاية كل سطر . فتنتهى معظم الأسطر بفاصلة منقوطة . وبعض الأسطر لاتنتهى بأى تنقيط . كما أن آخر سطر ينتهى بنقطة . وهذا هو جزء من تكوين لغة البسكال .

وسوف نتعرض للتنقيط بتوسع فى أقسام لاحقة من أقسام هذا الكتاب .

## ٦ - الخواص الجيدة للبرنامج 6 - DESIRABLE PROGRAM CHARACTERISTICS

دعنا نفحص بعض الخواص الهامة لبرامج الكمبيوتر المكتوبة بطريقة جيدة . وهذه الخواص تنطبق على أى برامج مكتوبة بأى لغة برمجة ، ولاتقتصر على البرامج المكتوبة بلغة البسكال . ويمكنها أن تقدم لنا مجموعة مفيدة من الخطوط الإرشادية فيما بعد فى هذا الكتاب عندما نبدأ فى كتابة برامج بلغة البسكال .

١ - تشير السلامة integrity إلى دقة الحسابات ، فيجب أن يكون واضحا أن كل التعزيزات الأخرى للبرنامج لامعنى لها إذا لم تنفذ الحسابات بطريقة صحيحة . وعلى هذا ... فإن سلامة الحسابات هى ضرورة حتمية فى أى برنامج من برامج الكمبيوتر .

٢ - ويشير الوضوح Clarity إلى إمكانية القراءة الشاملة للبرنامج ، مع التركيز الخاص على المنطق الذى يحتويه . فإذا ماكتب البرنامج بوضوح ، فيجب أن يكون فى مقدرة أى مبرمج آخر أن يتتبع منطق البرنامج ، بدون بذل مجهود . ( كما يجب أن يكون ممكنا لكاتب البرنامج أيضا أن يتتبع برنامجه بعد تركه للبرنامج فترة زمنية طويلة ) . وأحد أهداف التصميم فى لغة البسكال ، هو إعداد برامج واضحة ومقروءة من خلال منهج منظم للبرمجة .

٣ - البساطة Simplicity . عادة مايتم تحقيق وضوح البرنامج ودقته بجعل الأشياء سهلة بقدر الإمكان ، وجعلها متناسقة مع الأهداف الشاملة للبرنامج . وفى واقع الأمر ، قد يكون مرغوبا فيه التضحية بقدر معين من كفاءة الحسابات ، وذلك للحفاظ على البساطة النسبية ، وعلى هيكل مباشر للبرنامج .

٤ - وتهتم الكفاءة efficiency بسرعة التنفيذ وكفاءة استغلال الذاكرة . وهناك أهداف عامة ، بالرغم من أنه يجب عدم تحقيقها على حساب الوضوح والبساطة . تتطلب معظم البرامج المعقدة المساومة بين هذه الخواص . وفى مثل هذه المواقف تكون الخبرة والإحساس العام من العوامل الأساسية .

٥ - التجزئة modularity . معظم البرامج الكبيرة يمكن تجزئتها إلى عدة أنشطة جزئية محددة . ومن البرمجة العملية الجيدة ، تنفيذ كل من هذه الأنشطة الجزئية كبرنامج لجزء module منفصل . ( ويشار إلى مثل هذه الأجزاء فى البسكال بأنها إجراءات procedures أو دوال functions ) . ويعزز استخدام التجزئة فى البرمجة من دقة ووضوح البرنامج ، كما أنه يسهل من إجراء أى تعديلات مستقبلية على البرنامج .

٦ - العمومية generality . عادة مانريد أن يكون البرنامج عاما بقدر الإمكان ، وداخل حدود معقولة . فمثلا يمكننا أن نصمم برنامجًا لقراءة قيم مؤشرات parameters رئيسية معينة ، بدلا من وضع قيم ثابتة فى البرنامج . وكقاعدة عامة ... يمكن الحصول على كمية عمومية معتبرة ببذل مجهود إضافى بسيط فى البرمجة .

## أسـئلة للمراجـعة Review Questions

١ – ماذا يعنى جهاز الكمبيوتر الكبير ؟ وأين يمكن أن توجد مثل هذه الأجهزة ؟ وفى أى شئ تستخدم هذه الأجهزة بصفة عامة ؟

٢ – ماهو جهاز المينى كمبيوتر ؟ وماهو اختلاف أجهزة المينى كمبيوتر عن أجهزة الكمبيوتر الكبيرة ؟

٣ – ماهو جهاز الميكروكمبيوتر ؟ وماهو اختلاف أجهزة الميكروكمبيوتر عن أجهزة المينى كمبيوتر وأجهزة الكمبيوتر الكبيرة؟

٤ – اذكر نوعين مختلفين من البيانات .

٥ – ماذا يعنى برنامج الكمبيوتر ؟

٦ – ماذا يحدث بصفة عامة عند تنفيذ برنامج كمبيوتر ؟

٧ – ماهى ذاكرة الكمبيوتر ؟ ومانوع المعلومات التى تخزن فيها ؟

٨ – ماذا تعنى ( بت ) ؟ وماذا تعنى ( بايت ) ؟

٩ – ماهو الفرق بين ( البايت ) و ( الكلمة ) بالنسبة للذاكرة ؟

١٠– ماهى المصطلحات المستخدمة فى وصف ذاكرة الكمبيوتر ؟ وماهى بعض السعات المعتادة ؟

١١ – اذكر بعض وحدات الذاكرات الثانوية . وكيف تختلف هذه الذاكرات عن ذاكرة الكمبيوتر الرئيسية ؟

١٢ – ماهى وحدات الوقت المستخدمة للتعبير عن سرعة تنفيذ الأنشطة الفردية بواسطة الكمبيوتر ؟

١٣ – ماهو الفرق بين تشغيل الدفعة والتشغيل المتداخل ؟ وماهى مميزات وعيوب كل منهما ؟

١٤ – ماذا تعنى الشاشة المرئية Console ؟

١٥ – ماذا تعنى المشاركة الزمنية ؟ وأى نوع من أنوا ع التطبيقات تناسبه المشاركة الزمنية ؟

١٦ – ماهى لغات الآلة ؟ وماهو الفرق بينهما وبين اللغات مرتفعة المستوى ؟

١٧ – اذكر بعض اللغات مرتفعة المستوى شائعة الاستخدام .

١٨ – ماهى مميزات استخدام اللغات مرتفعة المستوى ؟

١٩ – ماذا تعنى الترجمة ؟ وماذا يعنى التفسير ؟ وكيف تختلف هاتان العمليتان عن بعضهما ؟

٢٠ – ماهو برنامج المصدر ؟ وماهو برنامج التشغيل ؟ وماسبب أهمية هذين المفهومين ؟

٢١ – ماذا تعنى البرمجة المرتبة ؟

٢٢ – من هو ( بليز بسكال ) ؟ ولماذا ذكر اسمه فى هذا الفصل ؟

٢٣ – من هو أول من طور لغة البسكال ؟ وماذا كان هدفه الأساسى ؟

٢٤ – لخص الحالة الحالية لقياسية لغة البسكال فى كل من أوروبا والولايات المتحدة الأمريكية .

٢٥ – ماهما المكونان الرئيسيان الذى يجب وجودهما فى كل برنامج مكتوب بلغة البسكال ؟ وماهو الغرض من كل منهما ؟

٢٦ – ماهما الجزآن الرئيسيان للمجموعة ؟ وماهو الغرض من كل منهما ؟

٢٧ – ماهى الكلمات المحجوزة أو الكلمات الرئيسية ؟

٢٨ – ماهى الدوال القياسية ؟

٢٩ – لماذا ترحل بعض العبارات داخل البرنامج المكتوب بلغة البسكال ؟

٣٠ – لخص معنى كل من خواص البرامج التالية : السلامة ، والوضوح ، والبساطة ، والكفاءة ، والتجزئة ، والعمومية . ماسبب أهمية كل خاصية من هذه الخواص ؟ .

## مشاكـــل Problems

٣١ – حدد الغرض من كل من البرامج المكتوبة بلغة البسكال التالية :

(A)
```
PROGRAM greeting(output);
BEGIN
   writeln('Welcome to the Wonderful');
   writeln;
   writeln('World of Computing!')
END.
```

(*b*)
```
PROGRAM payroll(input,output);
CONST rate = 0.14;
VAR gross,tax,net : real;
BEGIN
   read(gross);
   tax := rate*gross;
   net := gross-tax;
   write(gross,tax,net)
END.
```

(*c*)
```
PROGRAM order(input,output);
VAR a,b : integer;
BEGIN
   read(a,b);
   IF a <= b THEN write(a,b) ELSE write (b,a)
END.
```

٣٢ – حدد بالنسبة للبرامج المكتوبة بلغة البسكال فى المشكلة السابقة ، الغرض من كل سطر من أسطر كل برنامج من هذه البرامج ، وذلك على قدر استطاعتك .

## الفصل الثانى

# أساسيات البسكال

# Pascal Fundamentals

يهتم هذا الفصل بالعناصر الأساسية المستخدمة فى تكوين عبارات بسيطة بلغة البسكال . وتشمل هذه العناصر مجموعة رموز البسكال ، والكلمات المحجوزة ، والمعرفات ، والأرقام ، والسلاسل ، والثوابت ، والمتغيرات والتعبيرات . وسوف نرى كيف يمكن خلط هذه العناصر لتكوين عبارات بسيطة ، لكنها كاملة بلغة البسكال . ويجب توجيه بعض الانتباه أيضا إلى أنواع العبارات المختلفة المتاحة ، والعناصر التى تحتويها ، والأهداف منها .

بعض هذه المادة مشروح بتفصيل كبير ، وعلى هذا ... فقد توجد صعوبة بعض الشئ فى استيعابه ، خاصة بالنسبة للمبرمج الذى ليس لديه أى خبرة بالبرمجة . تذكر على أية حال أن الغرض الحالى من هذه المادة هو تقديم بعض المفاهيم الأساسية ، وبعض التعريفات اللازمة للمواضيع التى سوف تذكر فى الفصول القليلة التالية . وعلى ذلك ... فعند قراءة هذا الفصل لأول مرة ، فإنك لست فى حاجة إلا إلى اكتساب اعتيادا عاما بالمواضيع الفردية . وسوف تكتسب فهما أكثر تفصيلا من الإشارة المتكررة لهذه المادة ، والتى تحدث فى الفصول التالية .

## ١ – مجموعة رموز البسكال : 1. THE PASCAL CHARACTER SET

تستخدم لغة البسكال الحروف الهجائية من A إلى Z ( كل من الحروف الكبيرة والحروف الصغيرة ) ، والأرقام من 0 إلى 9 وبعض الرموز الخاصة ، وذلك فى بناء مجموعة من العناصر الأساسية للبرنامج ( الأرقام والمعرفات والتعبيرات وغيرها ) . وفيما يلى توجد الرموز الخاصة :

| | | | |
|---|---|---|---|
| + | . | < | ( |
| - | : | <= | ) |
| * | ; | > | [ |
| / | , | >= | ] |
| := | ' | <> | { |
| = | ^ | .. | } |

وتستخدم الرموز ( * and * ) فى بعض أجهزة الكمبيوتر ، بدلا من {and} . كما يمكن استخدام الرموز (. and .) ، بدلا من [ and ] . ويمكن أيضا أن يحل الرمز @ محل ^ فى بعض أجهزة الكمبيوتر .

لاحظ أن بعض هذه الرموز مكون من رمزين منفصلين متتالين ( مثل >= ، := و .. الخ ) .

## ٢ – الكلمات المحجوزة : 2. RESERVED WORDS

هناك كلمات محجوزة معينة ، لها معنى قياسى سبق تعريفه فى لغة البسكال . وفيما يلى هذه الكلمات :

| | | | |
|---|---|---|---|
| AND | END | NIL | SET |
| ARRAY | FILE | NOT | THEN |
| BEGIN | FOR | OF | TO |
| CASE | FUNCTION | OR | TYPE |
| CONST | GOTO | PACKED | UNTIL |
| DIV | IF | PROCEDURE | VAR |
| DO | IN | PROGRAM | WHILE |
| DOWNTO | LABEL | RECORD | WITH |
| ELSE | MOD | REPEAT | |

ويمكن استخدام هذه الكلمات المحجوزة للغرض المحدد لها فقط . ولايمكن أن يعرفها المبرمج أى تعريف اختيارى آخر . ومن المعتاد عرض الكلمات المحجوزة فى برنامج البسكال ، إما فى الحروف العلوية ، أو على شكل بارز . وسوف نستخدم الحروف العلوية خلال هذا الكتاب .

## ٣ - المعرفـــات : 3. IDENTIFIERS

المعرف Identifier هو اسم يعطى لأحد عناصر البرنامج ، مثل : الثابت ، أو المتغير ، أو الإجراء ، أو البرنامج . وتتكون المعرفات من حروف أو أرقام بأى ترتيب ، فيما عدا أول خانة ، والتى يجب أن تحتوى على حرف . ويسمح باستخدام الحروف الكبيرة والحروف الصغيرة ، وليس هناك تمييز للحروف الكبيرة يميزها عن الحروف الصغيرة . وسوف نستخدم على أية حال الحروف الصغيرة فى هذا الكتاب فى المعرفات ، وذلك لتمييزها عن الكلمات المحجوزة ( والتى تظهر بحروف كبيرة ) .

كما تسمح بعض صيغ البسكال ببعض رموز أخرى أيضا ، مثل الشرطة التى توضع تحت الحرف .

### مـثال (٢-١)

الأسماء التالية هى معرفات صحيحة :

| | | | |
|---|---|---|---|
| x | y12 | sum | temperature |
| names | area | taxrate | table1 |

والأسماء التالية هى معرفات غير صحيحة للأسباب المذكورة أمام كل منها :

c-max لايسمح باستخدام رموز غير الحروف والأرقام .

( فى بعض صيغ البسكال يمكن أن يكون هذا المعرف صحيحًا ) .

4th أول خانة يجب أن يشغلها حرف أبجدى .

array كلمة ARRAY هى كلمة محجوزة .

last word غير مسموح باستخدام فراغ ( تذكر أن الفراغ يعتبر رمزا ) .

ويمكن أن يكون للمعرف أى طول ، إلا أن بعض مترجمات البسكال لاتميز إلا أول 8 خانات فقط . وفى مثل هذه الحالة ، فإن بقية الخانات لاتوجد إلا لإظهار معنى معين للمبرمج .

### مـثال (٢-٢)

المعرفان filemanager و filemanagement هما معرفان صحيحان من ناحية القواعد . إلا أن الكمبيوتر قد لايستطيع أن يميز بينهما على أية حال ، بسبب أن أول ثمانى خانات تشغلها نفس الحروف فى كل من المعرفين . وعلى هذا ... فيجب ألا يستخدم إلا معرف واحد منهما فى نفس البرنامج ( إلا إذا كان معروفا أن مترجم البسكال المستخدم لايضع قيدا على طول المعرف ) .

وكقاعدة عامة ... يجب أن يحتوى المعرف على عدد كافٍ من الرموز ، بحيث يكون معناه واضحا . وفى الناحية الأخرى ، يجب تجنب استخدام عدد كبير من رموز لامعنى لها .

### مـثال (٢-٣)

فى برنامج بسكال مكتوب لحساب قيمة الاستثمار المستقبلية ، يمكن أن يكون لقيمة الاستثمار والقيمة المستقبلية أسماء رمزية مناسبة . وعلى أية حال ... فاستخدام v و fv ربما يكون موجزا جدا ، حيث إن الهدف من هذه المعرفات لايكون واضحا . وفى الناحية الأخرى ... فإن المعرف futurevalueofaninvestement يكون غير مقنع ، نظرا لأنه طويل ومرهـق .

## ٤ - المعرفات القياسية : 4. STANDARD IDENTIFIERS

تحتوى لغة البسكال على بعض المعرفات القياسية التى لها معان سبق تحديدها . وهذه المعرفات القياسية هى :

```
abs       false     pack      sin
arctan    get       page      sqr
boolean   input     pred      sqrt
char      integer   put       succ
chr       ln        read      text
cos       maxint    readln    true
dispose   new       real      trunc
eof       odd       reset     unpack
eoln      ord       rewrite   write
exp       output    round     writeln
```

وتحتوى بعض صيغ البسكال على معرفات قياسية أخرى إضافية . ويجب على القارئ أن يحدد بدقة المعرفات القياسية المتاحة فى صيغة اللغة التى يستخدمها .

وعلى عكس الكلمات المحجوزة ( والتى لايمكن إعادة تعريفها على الإطلاق ) ، يمكن للمبرمج أن يعيد تعريف المعرفات القياسية . ويتم عمل ذلك باستخدام إيضاحات وتعريفات مناسبة ، كما سيوضح ذلك فيما بعد فى هذا الكتاب . ومن البرمجة الجيدة فى معظم الأحوال على أية حال أن تستخدم المعرفات القياسية طبقا للغرض الذى سبق تحديده لها . وهذا صحيح بصفة خاصة بالنسبة للبرامج التى يكتبها المبرمجون المبتدئون . وعلى هذا ... فيجب معاملة المعرفات القياسية بنفس طريقة معاملة الكلمات المحجوزة .

## ٥ - الأعداد : 5. NUMBERS

يمكن كتابة الأعداد بعدة طرق مختلفة فى لغة البسكال . ويمكن أن يشتمل العدد على إشارة ، أو علامة عشرية ، أو أس ( أو معامل للقوة المرفوع لها الرقم scale factor ) اذا كانت هناك حاجة لذلك . والقواعد التالية تطبق مع كل الأعـداد .

(١) لايمكن أن يحتوى العدد على فاصلة أو على فراغ .

(٢) يمكن أن تسبق العدد إشارة موجب ( + ) ، أو إشارة سالب ( - ) إذا كانت هناك حاجة لذلك . فإذا لم توجد إشارة ، فيعتبر العدد موجبا .

(٣) لايمكن أن يتعدى العدد قيما معينة كحد أدنى وكحد أقصى . وتعتمد هذه القيم على كل من نوع العدد ، ونوع الكمبيوتر المستخدم ، وعلى نوع المترجم المستخدم أيضا .

## الأعداد الصحيحة : Integer Numbers

لايحتوى العدد الصحيح integer number على علامة عشرية ، أو على أس . وعلى هذا ... فإن العدد الصحيح يكون عبارة عن سلسلة من الأرقام ، تسبقها ( بصورة اختيارية ) إشارة موجب أو إشارة سالب .

### مثال (٢-٤)

الأعداد الصحيحة التالية هى أعداد صحيحة :

| | | | |
|---|---|---|---|
| 0 | 1 | +1 | -1 |
| 743 | -5280 | 60000000 | -999999 |

والأعداد الصحيحة التالية غير صحيحة طبقا للأسباب المذكورة أمام كل منها :

123,456 ممنوع استخدام الفواصل فى العدد الصحيح .

36. ممنوع استخدام العلامة العشرية فى العدد الصحيح .

10 20 30 ممنوع استخدام الفراغات فى العدد الصحيح .

ويمكن أن تتراوح قيمة العدد الصحيح من صفر إلى قيمة قصوى تختلف قيمتها من جهاز كمبيوتر لجهاز آخر ، ومن مترجم لمترجم آخر . وهناك قيمة قصوى معتادة لأجهزة الميكروكمبيوتر ، وهى 32767 ، كما أن بعض أجهزة الكمبيوتر الأخرى تسمح بأعداد صحيحة أكبر من هذا العدد كثيرا . ( وتعرف القيمة القصوى باستخدام المعرف القياسى maxint كما هو موضح فيما بعد ) . ويجب أن يحدد القارئ أقصى قيمة متاحة لدى الكمبيوتر الذى يستخدمه .

## الأعداد الحقيقية Real Numbers

يجب أن يحتوى العدد الحقيقى real number على علامة عشرية ، أو على أس ( أو كليهما ) . فإذا مااستخدمت علامة عشرية مع العدد ، فيجب أن تظهر بين رقمين . وعلى هذا ... فلايمكن أن يبدأ أو ينتهى العدد الحقيقى بعلامة عشرية .

### مثال (٢-٥)

الأعداد الحقيقة التالية هى أعداد صحيحة :

| | | | |
|---|---|---|---|
| 0.0 | 1.0 | -0.2 | 827.602 |
| 50000.0 | -0.000743 | 12.3 | -315.0066 |

والأعداد الحقيقية التالية هى أعداد غير صحيحة ، طبقا للأسباب المكتوبة أمام كل منها :

1. يجب ألا ينتهى العدد الحقيقى بعلامة عشرية .

1,000.0 ممنوع استخدام الفواصل .

.333333 يجب ألا يبدأ العدد الحقيقى بعلامة عشرية .

50 يجب أن تظهر علامة عشرية أو أس فى العدد الحقيقى .

ويمكن أن يحتوى العدد الحقيقى على أس ، وذلك لترحيل موقع العلامة العشرية . ( إذا لم توجد علامة عشرية فى العدد ، فيفترض أنها موجودة على يمين آخر رقم ) . وهذا مثل التمثيل العلمى بالضرورة ، فيما عدا أن الأساس 10 يستبدل بالحرف E أو e . وعلى هذا ... فالعدد $1.2 \times 10^{-3}$ يكتب بالبسكال على الصورة 1.2E-3 ، أو الصورة 1.2e-3 . ويجب أن يكون الأس نفسه رقما صحيحا موجبا أو سالبا .

## مـثال (٢-٦)

يمكن تمثيل الكمية $3 \times 10^{10}$ بلغة البسكال فى إحدى الصيغ التالية

```
3.0E+10      3.0E10      3e+10      3E10
0.3E+11      0.3e11      30.0E+9    30e9
```

وبالمثل يمكن تمثيل الكمية $-5.026 \times 10^{-17}$ بلغة البسكال بإحدى الصيغ التالية :

```
-5.026E-17    -0.5026E-16    -50.26e-18    -0.0005026e-13
```

## مـثال (٢-٧)

الأعداد الحقيقية التالية هى أعداد صحيحة :

```
2E-8      -0.006e-5      1.6667E+8      +0.12121212e12
```

والأعداد الحقيقية التالية خاطئة طبقا للأسباب المذكورة أمام كل منها :

3.E+10 يجب أن يظهر رقم فى كل ناحية من نواحى العلامة العشرية .

8e2.3 الأس يجب أن يكون صحيحا .

.3333e-3 يجب أن يظهر رقم قبل العلامة العشرية .

3E 10 ممنوع ظهور الفراغات فى الرقم .

والأعداد الحقيقة لها مدى أكبر كثيرا من الأعداد الصحيحة . وعادة ماتتراوح قيمة الأعداد الحقيقية من أقل قيمة تعادل 1E-38 إلى أقصى قيمة تعادل 1E+38 . ( وتتغير هذه القيم من جهاز كمبيوتر لجهاز آخر ، ومن مترجم

بسكال لمترجم آخر ) . وبالإضافة إلى ذلك ... فإن الرقم 0.0 ، وهو أقل من 1E-38 ، هو رقم صحيح أيضا . ويجب أن يحدد القارئ القيم المناسبة للجهاز المتاح له .

ويتغير عدد الأرقام المعنوية في العدد الحقيقي من إحدى صيغ لغة البسكال لصيغة أخرى . وتسمح معظم الصيغ بسبعة أو ثمانية أرقام معنوية ، والتي تكفي بالنسبة لمعظم التطبيقات . ويجب أن يحدد القارئ عدد الأرقام الصحيحة المعنوية المناسبة للجهاز الذي يستخدمه .

## الدقة العددية : Numerical Precision

يجب أن يكون مفهوما أن الأعداد الصحيحة هي كميات مضبوطة ، بينما الأعداد الحقيقية هي تقريبات . وعلى هذا ... فإن العدد الحقيقي 1.0 يمكن أن يكون ممثلا داخل ذاكرة الكمبيوتر على أنه ......0.99999999 ، بالرغم من أنه يمكن أن يظهر على أنه 1.0 على الشاشة ، أو إذا ما تمت طباعته . وعلى هذا ... فلايمكن استخدام الأعداد الحقيقية في أغراض خاصة ، مثل العد ، أو عمل الفهارس ، وما إلى ذلك ، حيث تكون القيم الدقيقة مطلوبة . وسوف نناقش هذه القيود كلما دعت الحاجة لذلك في فصول أخرى من هذا الكتاب .

## ٦ - السلاسل : 6. STRINGS

السلسلة string هي تسلسل من الرموز ( مثل الحروف والأرقام والرموز الخاصة ) ، وتكون موضوعة بين علامتي تنصيص .

### مثال (٢-٨)

السلاسل التالية هي سلاسل صحيحة :

```
'GREEN'        'Washington, D.C. 20005'     '270-32-3456'

'$19.95'       'THE CORRECT ANSWER IS:'     '2*(I+3)/J'
```

وأقصى عدد للرموز التي يمكن أن توجد بين علامتي التنصيص يتغير من إحدى صيغ البسكال لصيغة أخرى . ومعظم الصيغ تسمح بأن يكون أقصى طول للسلسلة 256 رمزا . وهو طول كاف لمعظم الأغراض .

وإذا ما احتوت السلسلة على علامتي تنصيص ، فيجب أن تمثل علامة التنصيص بعلامتي تنصيص متتاليتين . فإذا ماحدث ذلك ، فلن يظهر في الطباعة سوى علامة تنصيص واحدة . وعلى ذلك ... فتفسر علامة التنصيص الواحدة بأنها محدد للسلسلة string delimiter ، بينما تفسر علامتا التنصيص المتتاليتان كعلامة تنصيص موجودة داخل السلسلة .

### مثال (٢-٩)

يمكن أن يحتوي برنامج بسكال على السلسلة التالية :

```
'PLEASE DON''T VERB YOUR NOUNS'
```

( لاحظ علامة التنصيص المتكررة فى الكلمة "don't." ) ، فإذا ماطبع البرنامج هذه السلسلة ، فإنها تظهر على النحو التالى :

```
PLEASE DON'T VERB YOUR NOUNS
```

وعادة ماتستخدم السلاسل فى كتابة عبارات كعناوين للمخرجات . وسوف نناقش ذلك بالتفصيل فى الفصل الرابع من الكتاب .

## ٧ - أنواع البيانات : 7. DATA TYPES

أحد الخواص الهامة والممتعة للغة البسكال هو مقدرتها على دعم العديد من أنواع البيانات المختلفة . ويشمل هذا بيانات من نوع بسيط ، وبيانات من نوع مركب ، وبيانات من نوع مشير .

والبيانات من النوع البسيط Simple-type data هى عناصر فردية ( أعداد أو رموز .. الخ ) ، وتصاحبها معرفات فردية على أساس معرف لكل منها . وفى واقع الأمر ... فإنه هناك أنواع عديدة من البيانات البسيطة . وهذه تشمل الأربعة أنواع القياسية للبيانات ، وهى : بيانات صحيحة ، وبيانات حقيقية ، وبيانات حرفية ، وبيانات بوليان ، والأنواع البسيطة التى يعرفها المستفيد والتى تشمل النوع المتعدد ونوع المدى الجزئى . وأنواع البيانات القياسية سوف تناقش بالتفصيل فى الفصل القادم . أما الأنواع البسيطة التى يعرفها المستفيد ، فسوف تؤخذ فى الاعتبار فى الفصل الثامن من الكتاب .

والبيانات من النوع المرتب structured-type data تحتوى على عناصر بيانات متعددة ، وترتبط عنصرا بعنصر بنفس الطريقة المحددة لذلك . ويصاحب كل مجموعة من عناصر البيانات معرف خاص . ويمكن أن يصاحب كل عنصر بيان فردى موجود داخل معرف خاص مناظر له . وهناك أربعة أنواع من البيانات المرتبة فى البسكال ، وهى المنظومات والسجلات والملفات والفئات .

والبيانات من النوع المشير pointer-type data تستخدم لتكوين بيانات ديناميكية من النوع المركب . والوصف البسيط لخواصها واستخداماتها يقع فى مدى خارج المناقشة الحالية ( انظر الفصل الثالث عشر من الكتاب ) .

وأنواع البيانات المختلفة ملخصة أدناه :

### (أ) بيانات بسيطة :

*(أ) بيانات قياسية :*

١) صحيح .

٢) حقيقى .

٣) حرفى .

٤) بوليان .

*(ب) بيانات يعرفها المستفيد :*

١) متعددة .

٢) مدى جزئى .

(٢) بيانات مرتبة :

(أ) منظومات .

(ب) سجلات .

(ج) ملفات .

(د) فئات .

(٣) بيانات مشيرة .

وسوف نهتم حاليا بالبيانات البسيطة فقط ، حيث ستناقش البيانات المركبة والبيانات المشيرة فى فصول لاحقة من الكتاب .

## ٨ - الثوابت : 8. CONSTANTS

من المقنع عادة تتبع عنصر بيانات بسيط ، مثل القيمة العددية أو السلسلة ، والذى يكون له معرف ، والذى يقدم أسمًا لعنصر البيانات . ويسمى المعرف ثابتا constant إذا ماحُدد عنصر البيانات بصورة دائمة ( أى إذا كانت قيمة عنصر البيانات غير متغيرة طوال فترة تنفيذ البرنامج )

ويجب أن يعرف الثابت دائما قبل أن يظهر فى أى عبارة من عبارات البسكال . ويخدم هذا التعريف هدفين ، حيث إنه يحدد أن المشير ثابت ، كما أنه يصاحب قيمة الثابت . وسوف يقوم عنصر البيانات بتحديد الثابت داخليا .

والصيغة العامة لتعريف الثابت تكتب على النحو التالى :

```
CONST name = value
```

حيث name هو معرف يمثل اسم الثابت ، و Value هى عنصر البيانات الفعلى الذى يحدد له المعرف name .

### مثال (٢-١٠)

تطلب أحد برامج البسكال استخداما متكررا للقيمة العددية 0.1666667. وعلى هذا ... فقد يكون من المقنع إدخال ثابت اسمه fraction ليستخدم فى مكان القيمة العددية الحقيقية . ويمكن تعريف هذا الثابت بكتابة مايلى :

```
CONST fraction = 0.1666667;
```

والإشارة التالية للمعرف fraction تكون مكافئة للإشارة إلى القيمة العددية الحقيقية . لاحظ أن fraction يعتبر ثابتًا حقيقيًا ، حيث إنه يصاحبه عدد حقيقى .

### مثال (٢-١١)

افرض أن أحد برامج البسكال يتكرر فيه استخدام السلسلة التالية :

```
'The Super-Duper Computer Company'
```

وذلك فى كتابة عناوين بأحد التقارير . ويمكن تمثيل هذه السلسلة بسهولة كثابت اسمه title . ولعمل ذلك ، فإننا نكتب مايلى :

```
CONST title = 'The Super-Duper Computer Company';
```

وذلك فى بداية البرنامج ... فإذا ماأردنا أن نطبع السلسلة الفعلية فى نقطة لاحقة من نقاط البرنامج ، فإننا نشير إلى المعرف title فى عبارة المخرجات المناسبة . ( سوف نناقش عمليات المدخلات والمخرجات فى الفصل الرابع من الكتاب ) .

ويعتبر title فى هذا المثال ثابتا من نوع السلسلة ، حيث إنه هناك سلسلة مصاحبة له .

## ٩ - المتغيرات : 9. VARIABLES

المعرف الذى يسمح لقيمته بالتغير أثناء تنفيذ البرنامج يسمى متغيراً Variable . ويجب توضيح كل متغير على حدة ، أى يجب تعريف كل متغير على حدة ، وذلك قبل أن يظهر فى البرنامج . ويحدد توضيح المتغير الحقيقى بأن المعرف هو متغير ( بدلا من أن يكون ثابتا أو غير ذلك ) كما يحدد نوع المتغير . وعلى عكس تعريف الثابت ، فلايكون هناك عنصر بيانات ( أى قيمة عددية أو سلسلة ) مصاحبة للمتغير ، وذلك فى توضيح أو تعريف المتغير .

والصيغة العامة لتوضيح المتغير هى :

```
VAR name : type
```

أو إذا كان هناك متغيرات متعددة من نفس النوع ، فإن الصيغة العامة تصبح :

```
VAR name 1, name 2, . . . , name n : type
```

حيث إن name 1 و name 2 .. الخ هى معرفات تمثل أسماء فردية لمتغيرات . وتشير type إلى نوع بيانات المتغير .

### مثال (٢-١٢)

يحتوى أحد برامج البسكال على متغيرات صحيحة ، اسمها row و column ، ومتغير حقيقى اسمه value ، ومتغير حرفى اسمه flag . وعلى هذا ... فيمكن أن يحتوى البرنامج على التوضيحات التالية :

```
VAR row,column : integer;
    value : real;
    flag : char;
```

وعنصر البيانات المختار يمكن أن يحدد بأنه ينتمى إلى أحد هذه المتغيرات ، وذلك فيما بعد فى البرنامج .

### مثال (٢-١٣)

افرض أن البرنامج يحتوى الآن على الثوابت الموصوفة فى المثال رقم ٢ -١٠، والمثال رقم ٢ -١١، والمتغيرات الموصوفة فى المثال رقم ٢ -١٢. فى هذه الحالة تصبح قائمة التوضيحات الكاملة على النحو التالى :

```
CONST fraction = 0.1666667;
      title = 'The Super-Duper Computer Company';
VAR row,column : integer;
    value : real;
    flag : char;
```

تذكر أنه يجب أن تظهر التوضحيات والتعريفات فى ترتيب معين ، وأنه يجب أن تسبق تعريفات الثوابت لتوضيحات المتغيرات .

## 10. EXPRESSIONS

## ١٠ - التعبيرات :

التعبير expression هو عبارة عن تجميع من العناصر لعناصر operands ( أى الأرقام أو الثوابت أو المتغيرات .. الخ ) متصلة مع بعضها بواسطة مؤثرات Operators لتكوين اصطلاح جبرى يمثل قيمة ( مثل عنصر بيانات بسيط ) . وهناك نوعان من التعبيرات فى البسكال، وهى : تعبيرات عددية ، وتعبيرات بوليان . والتعبير العددى numerical expression يمثل قيمة عددية ، بينما يمثل التعبير البوليانى boolean expression شرطا منطقيا ، تكون قيمته صحيحة أو خاطئة فقط .

### مثال (٢-١٤)

فيما يلى مثالا لتعبير عددى

```
(b*b-4*a*c)/(2*a)
```

وتسمى المعرفات a و b و c والأرقام 4 و 2 بأنها العناصر التى يؤثر عليها operands ، بينما تسمى الرموز * و - و / بأنها المؤثرات operators المناظرة ( والتى تمثل الضرب والطرح والقسمة على التوالى ) . وتستخدم الأقواس لتحدد ترتيب تنفيذ العمليات . ويمثل التعبير عددا معينا . وعلى هذا ... فإذا ماكانت a و b و c تمثل القيم 1 و 2 و 3 على التوالى ، فإن هذا يعنى أن التعبير يمثل القيمة 4- .

وعند تكوين تعبير عددى ، يجب التمييز بين الكميات الصحيحة والكميات الحقيقية . وهذا صحيح بالنسبة لكل من العناصر التى يؤثر عليها ، وكذلك التعبير نفسه . وسوف نتعرض للمزيد عن ذلك فى الفصل التالى .

### مثال (٢-١٥)

فيما يلى مثالا لتعبير بوليانى

```
pay < 1000.0
```

فى هذا التعبير يكون pay متغيرا من النوع الحقيقى ، حيث إن 1000.0 هى عدد حقيقى ، والرمز < هو مؤثر بوليانى .لاحظ أن كل من pay و 1000.0 هى عناصر يؤثر عليها فى هذا التعبير البوليانى ) . وتكون قيمة التعبير صحيحة اذا مامثلت pay قيمة أقل من 1000.0 . وعلى العكس من ذلك ... تصبح قيمة التعبير خاطئة إذا مامثلت pay قيمة أكبر من 1000.0 أو تساويها .

وتستخدم التعبيرات البوليانية فى العديد من تكوينات التحكم ، مثل مكون IF - THEN التالى :

```
IF pay < 1000.0 THEN writeln(employeenumber);
```

وسوف يتسبب هذا المكون فى كتابة قيمة المتغير employeenumber إذا ماكانت قيمة pay أقل من 1000.0 .

وسوف نناقش استخدام تعبيرات بوليان داخل تكوينات التحكم بتفاصيل أكثر فى الفصل السادس من الكتاب .

ويجب على كل التعبيرات أن تتبع الشروط العامة التالية :

(١) غير مسموح بكتابة مؤثرين تاليين لبعضهما مباشرة . وعلى أية حال ... يمكن استخدام الأقواس لتفصل بين أى مؤثرين تاليين . ( تذكر أنه يجب أن تستخدم الأقواس أزواجا بصفة دائمة ) .

(٢) يمكن أن يحتوى التعبير على معرف واحد ، يستخدم كثابت أو كمتغير .

(٣) اسم الدالة ( أى دليل الدالة ) يمكن استخدامه فى مكان معرف لثابت ، أو فى مكان معـرف لمتغـير داخـل التعـبير ( سوف يذكر المزيد عن ذلك فيما بعد فى هذا الفصل ) .

## ١١ - العبارات : 11. STATEMENTS

عبارة statement البسكال هى عبارة عن أحد التعليمات ، أو مجموعة من التعليمات التى تجعل الكمبيوتر يتخذ إجراءً معيناً . وهناك نوعان أساسيان من العبارات فى البسكال : وهما العبارات البسيطة ، والعبارات المرتبة والعبارات البسيطة simple هى عبارات فردية بالضرورة ، وهى تعليمات غير شرطية تنفذ أحد الأنشطة التالية :

(١) تحدد عنصر بيانات لأحد المتغيرات ( وتسمى هذه العبارة بعبارة تحديد assignment ) .

(٢) توصل إجراء حسابى ذاتيا ، ويسمى إجراء procedure .

(٣) تنقل التحكم فى البرنامج غير المشروط إلى جزء آخر من أجزاء البرنامج ( عبارة GOTO ) .

### مـثال (٢-١٦)

فيما يلى عبارة تحديد

```
tax := 0.14*gross;
```

فى هذا المثال يفترض أن tax و gross متغيران من النوع الحقيقى ، وأن القيمة الحقيقية قد تحددت للمتغير gross . ( يمكن أن تكون هذه القيمة سبق قراءتها فى الكمبيوتر ، أو أنه سبق حسابها فى البرنامج ) . وتتسبب عبارة التحديد فى ضرب قيمة gross فى 0.14 ، وتحديد أن حاصل الضرب هو قيمة للمتغير tax .

لاحظ أن الرمز المستخدم للتحديد هو ( =: ) وليس = كما فى حالة معظم لغات البرمجة الأخرى . لاحظ أيضا أن طرف العبارة الأيمن ( وهو gross *0.14 ) عبارة عن تعبير عددى كما سبق وصف التعبير العددى .

### مـثال (٢-١٧)

فيما يلى عبارة من عبارات GOTO

```
GOTO 100;
```

ولايشجع استخدام عبارات GOTO فى البسكال . وسوف نذكر الكثير عن ذلك فى الفصل السادس من الكـتاب .

سوف نذكر الكثير عن العبارات البسيطة التى تصل الإجراءات بعضها ببعض ، وذلك فى القسم التالى .

وتميز لغة البسكال أنواعا عديدة من العبارات المرتبة . وهى تشمل مايلى :

(١) عبارات مركبة تحتوى على تسلسل من عبارتين أو أكثر ، متتالية وراء بعضها .

(٢) عبارات متكررة تشمل إعادة تنفيذ لعدة عبارات بسيطة .

(٣) عبارات شرطية تنفذ عبارة بسيطة واحدة أو أكثر ، إذا ماتحقق حدوث شرط منطقى محدد فقط .

### مثال (٢-١٨)

فيما يلي عبارة مركبة مأخوذة من مثال رقم ١ - ٧ من الفصل الأول .

```
BEGIN
   read(radius);
   area := 3.14159*sqr(radius);
   write(radius,area)
END
```

لاحظ أن العبارات البسيطة التي تتكون منها العبارة المركبة محصورة بين الكلمتين BEGIN و END . لاحظ أيضا أن العبارات البسيطة مفصولة عن بعضها باستخدام فواصل منقوطة .

### مثال (٢-١٩)

فيما يلي عبارة متكررة :

```
FOR count := 1 TO 100 DO write(count);
```

سوف تنفذ هذه العبارة 100 مرة . وفي كل مرة تنفذ فيها العبارة تطبع قيمة المتغير Count ، أو تظهر على الشاشة . وعلى هذا ... تتسبب العبارة في ظهور القيم التالية على وحدة المخرجات :

```
1  2  3  . . .  100
```

### مثال (٢-٢٠)

فيما يلي عبارة شرطية :

```
IF pay < 1000.0 THEN write('group 1') ELSE write('group 2');
```

تتسبب هذه العبارة في ظهور group1 في صورة المخرجات إذا ماكان المتغير pay يمثل قيمة أقل من 1000.0 . أما إذا كانت قيمة pay أكبر من 1000.0 أو تساويها . فتظهر group 2 . بدلا من ظهور group 1 .

وسوف تناقش العبارات المرتبة بتفاصيل أكبر في الفصل السادس من الكتاب . ويجب أن يهتم القارئ الآن بالنظرة العامة على المفاهيم العامة فقط .

## ١٢ - الإجراءات والدوال : 12.PROCEDURES AND FUNCTIONS

الإجراءات والدول هي عناصر محتواه ذاتيا في البرنامج ، وأحيانا مايشار إليها بأنها أجزاء modules تنفذ إجراءات محددة . ويمكن الاتصال بهذه الإجراءات من أي مكان في البرنامج . فإذا ما تم الاتصال بنفس الإجراء من عدة نقاط مختلفة في البرنامج ، فيمكن إعطاء هذا الإجراء معلومات مختلفة ( أي قيم مختلفة لعناصر البيانات المطلوبة ) عند كل نقطة من نقاط الاتصال به .

وعند الاتصال بأحد الإجراءات ، فإن المعلومات المقدمة يتم تشغيلها بواسطة العبارات الموجودة في الإجراء . وعادة مايتسبب ذلك في إنتاج معلومات جديدة . وعند ذلك تسود المعلومات إلى النقطة التي حدث اتصال فيها آخر مرة بالإجراء ، ويستمر البرنامج في التنفيذ عند هذه النقطة .

وتقدم المعلومات التي تمر إلى الإجراء كقائمة بعناصر البيانات ( أي كثوابت ومتغيرات وتعبيرات .. الخ ) ، والتي تسمى بالمؤشرات parameters . وتفصل هذه المؤشرات عن بعضها بواسطة فواصل ، كما أنها توضع كلها بين قوسين ، وذلك بعد اسم الإجراء مباشرة . ويمكن استخدام مؤشرات محددة في تمثيل معلومات جديدة يتم إنتاجها داخل الإجراء . وعلى هذا ... فيمكن أن تمثل المؤشرات معلومات يعيدها الإجراء أو معلومات تقدم للإجراء .

وتدعم لغة البسكال كل من الإجراءات والدوال القياسية التى يعرفها المستفيد . وسوف نركز على الإجراءات والدوال القياسية حاليا . وعناصر البرنامج هذه تكون موجودة فى مكتبة البسكال ، وهى جزء من اللغة . وسوف تقدم مناقشة كاملة للإجراءات والدوال التى يعدها المستفيد فى الفصل السابع من الكتاب .

وجميع الإجراءات procedures لها الخواص العامة التالية :

(١) يتم الوصول إلى الإجراء عن طريق عبارة بسيطة تحتوى على اسم الإجراء يتبعه قائمة ( اختيارية ) بالمؤشرات .

(٢) يمكن أن تمثل المؤشرات معلومات تقدم إلى الإجراء ، أو تحت ظروف معينة يمكن أن تمثل معلومات تعود من الإجراء .

(٣) يمكن نقل أى عدد من عناصر البيانات بين الإجراء والنقاط التى تشير إليه ( أى العبارة التى تصل إلى الإجراء ) .

مـثال (٢-٢١)

عادة ماتكون عبارة write دليلا ( أى تصل إلى إجراء ) لإجراء بسكال قياسى . وعلى هذا فإن العبارة التالية :

```
write(a,b,c);
```

ستسبب فى طباعة قيم المؤشرات a و b و c ، أو فى إظهارها على الشاشة .

لاحظ أن كل المؤشرات فى هذا المثال معلومات تقدم إلى الإجراء ، ولاتوجد أى معلومات جديدة تعود منه .

افرض أن برنامج البسكال يحتوى على عبارتى الكتابة write التاليتين :

```
write(a,b,c);
      .
      .
      .
write(x,y,z);
```

تتسبب أول عبارة فى طباعة قيم المتغيرات a و b و c ، بينما تقوم العبارة الثانية ( والمتطابقة تماما مع العبارة الأولى باستثناء المؤشرات ) بإخراج قيم المتغيرات z و y و x .

وجميع الدوال functions لها الخواص العامة التالية :

(١) يتم الاتصال بالدالة عن طريق تحديد اسمها فى أحد التعبيرات ، كما لو كانت متغيـراً عادياً ، ويتبـع اسـمها قائمـة ( اختيارية ) من المؤشرات .

(٢) يمكن أن يتبع اسم الدالة قائمة ( اختيارية ) من المؤشرات . وتستخدم هذه المؤشرات فى نقل معلومات إلى الدالة من نقطة الإشارة إليها فقط .

(٣) تقوم الدالة بإعادة عنصر بيانات واحد . وسوف يقوم اسم الدالة نفسه بتمثيل عنصر البيانات هذا .

(٤) يجب أن تكون الدالة من نفس نوع البيانات للتعبير الذى يقوم بالاتصال بها .

## مــثال (٢-٢٢)

اعتبر مرة أخرى العبارة التالية :

```
area := 3.14159*sqr(radius);
```

الطرف الأيمن عبارة عن تعبير عددى ، تمر من خلاله قيمة المتغير radius ، وذلك الى الدالة القياسية sqr ، وتعود هذه الدالة بقيمة مربع نصف القطر . وعند ذلك تضرب قيمة مربع نصف القطر فى 3.14159 ، وتحدد النتيجة للمتغير area .

لاحظ أن radius هو مؤشر يمثل عنصر بيانات يقدم إلى الدالة القياسية . أما عنصر البيانات الذى يعود من الدالة ، فيمثله اسم الدالة sqr . لاحظ أيضا أن نوع الدالة عددى ( حقيقية بصفة محددة ) ، وهذا ضرورى إذا ماكانت الدالة تستخدم فى تعبير عددى .

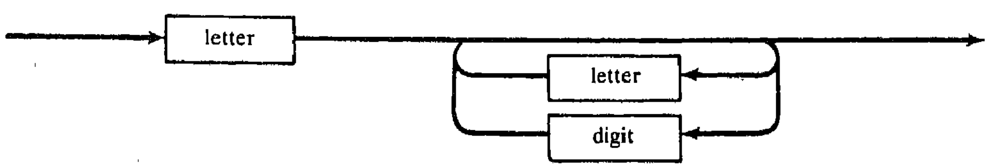
وسوف نذكر المزيد عن الاجراءات والدوال فى الفصل السابع من الكتاب . ولانحتاج حاليا إلا إلى الاهتمام بالمعلومات العامة التى سبق ذكرها عن هذا الموضوع . وسوف تسمح لنا هذه المعلومات بفهم المادة المعروضة فى الفصول القادمة .

## ١٣ – الرسومات التكوينية فى لغة البسكال : 13. PASCAL SYNTAX DIAGRAMS

من المناسب قبل ترك هذا الفصل أن تناقش طريقة شائعة الاستخدام فى تمثيل الأشكال التكوينية Syntactical ( أى الخاصة بقواعد اللغة ) فى لغة البسكال . وهى تشمل استخدام رسومات تكوينية ، مثل الرسم الموجود فى المثال التالى :

## مــثال (٢-٢٣)

يوضح الشكل ٢ – ١ رسما تكوينيا من الرسومات التكوينية للغة البسكال ، والذى يوضح الطريقة التى يمكن تمثيل المعرف بها . ويبين هذا الشكل أن المعرف يجب أن يبدأ بحرف كما هو موضح فى المستطيل الموجود على أقصى يسار الشكل ، والموجود به كلمة letter . ويلى هذا المستطيل مسار مستقيم له دورتان اختياريتان للعودة . تحتوى كل دورة على مستطيل يمثل نوع الرموز التى يمكن تواجدها كجزء من أجزاء اسم المعرف . وحيث إن المستطيل العلوى يحتوى على كلمة letter ، والمستطيل السفلى يحتوى على كلمة digit ، فإننا نستخلص من ذلك أنه يجب أن يبدأ أى معرف بحرف ، ويتبعه أى عدد من الحروف والأرقام .




الشكل ٢ – ١

ويمكن أن تصبح الرسومات التكوينية هذه معقدة جدا ، حتى بالنسبة للتكوينات البسيطة . وعلى هذا ... فمعظم المبتدئين يفضلون عدم استخدامها . وبعد أن يكتسب القارئ بعض الفهم والاعتياد على لغة البسكال ، فإن مثل هذه الرسومات تبدو أكثر فهما له .

توجد قائمة كاملة بكل الرسومات التكوينية فى لغة البسكال فى ملحق F . ونوصى القارئ بأن يعود إلى هذه الرسومات بصفة متكررة .

## أسئلة للمراجعة : Review Questions

(١) ماهى محتويات فئة رموز البسكال ؟

(٢) ماهى الكلمات المحجوزة فى البسكال ؟ وكيف يمكن تعريف الكلمات المحجوزة داخل هذا الكتاب ؟ .

(٣) ماهى معرفات البسكال ؟ وماهى مكونات المعرف ؟

(٤) كيف يمكن أن يحتوى المعرف على العديد من الرموز ؟ وهل كل هذه الرموز متساوية المعنى ؟

(٥) ماذا يعنى المعرف القياسى ؟

(٦) ماهو الفرق بين المفردات القياسية والكلمات المحجوزة ؟

(٧) لخص القواعد العامة لكتابة الأعداد فى البسكال .

(٨) ماهى القواعد الخاصة بالأعداد الصحيحة ؟

(٩) كيف يمكن تحديد أكبر عدد صحيح مسموح به لكل كمبيوتر ؟

(١٠) أذكر طريقتين مختلفتين لكتابة الأعداد الحقيقية . لخص القواعد التى تطبق فى كل حالة من هاتين الحالتين .

(١١) ماهو الغرض من الأس الموجود فى العدد الحقيقى ؟

(١٢) اذكر الفرق بين الأعداد الحقيقية والأعداد الصحيحة . تحت أى ظروف يجب استخدام كل نوع من هذين النوعين للأعداد ؟

(١٣) ماهى السلسلة ؟ وماهى الرموز التى يمكن أن تظهر فى السلسلة ؟

(١٤) ماهى القيود الموضوعة على أقصى طول السلسلة ؟

(١٥) كيف يمكن طباعة علامة تنصيص داخل إحدى السلاسل ؟

(١٦) اذكر ثلاثة أسماء مختلفة للبيانات المستخدمة فى البسكال .

(١٧) مامعنى البيانات بسيطة النوع ؟

(١٨) ماهى الأربعة أنواع القياسية للبيانات البسيطة ؟

(١٩) مامعنى النوع البسيط الذى يعرفه المستفيد ؟ وماهو الفرق بين هذا النوع من البيانات وأنواع البيانات القياسية ؟

(٢٠) مامعنى البيانات مرتبة النوع ؟ وماهو الفرق بينها وبين البيانات بسيطة النوع ؟

(٢١) اذكر أربعة أنواع مختلفة من البيانات المرتبة .

(٢٢) مامعنى الثابت ؟ وماذا يجب أن يحدث قبل أن يظهر الثابت فى عبارة بسكال ؟

(٢٣) كيف يحدد نوع الثابت ؟

(٢٤) مامعنى المتغير ؟ وماهو الفرق بينه وبين الثابت ؟

(٢٥) لخص القواعد المستخدمة فى توضيح أنواع البيانات للمتغيرات .

(٢٦) ماهو التعبير ؟ وماهى التعبيرات المستخدمة فى البسكال ؟ وماهو الاختلاف بينها ؟

(٢٧) ماهو العنصر المؤثر عليه ؟ وماهو المؤثر ؟ وكيف يستخدم كل منهما فى التعبير ؟

(٢٨) لخص الشروط العامة التى يجب أن تحققها كل التعبيرات .

(٢٩) ماهى العبارة ؟ وماهما النوعان الأساسيان للعبارات فى البسكال ؟ وماهو الفرق بينهما ؟ .

(٣٠) اذكر ثلاثة أنشطة مختلفة يمكن أن تؤديها العبارة البسيطة ؟

(٣١) ماهو الغرض من عبارة التحديد ؟ وماهو نوع هذه العبارة ؟

(٣٢) اذكر ثلاثة أنواع مختلفة للعبارات المرتبة .

(٣٣) ماهو الإجراء ؟ وماهى الدالة ؟ وماهو الفرق بين كل منهما ؟ .

(٣٤) ماهو الجزء module ؟ وماهى العلاقة بينه وبين الإجراء procedure والدالة function ؟

(٣٥) ماذا يحدث عند الاتصال بإجراء ؟

(٣٦) مامعنى مؤشرات ؟ وماهو الغرض منها ؟

(٣٧) اذكر فئتين عامتين للإجراءات والدوال .

(٣٨) لخص الخواص العامة للإجراءات .

(٣٩) لخص الخواص العامة للدوال .

(٤٠) ماهى دلالة اسم الدالة ؟ وماهو الفرق بينها وبين اسم الإجراء ؟

(٤١) مامعنى الشكل التكوينى ؟ وكيف تمثل هذه الأشكال فى البسكال ؟

## مسائل محلولة : Solved Problems

(٤٢) حدد : أى من المعرفات التالية صحيح . وإذا كان خاطئا ؛ حدد سبب الخطأ :

record-number خطأ – ممنوع ظهور أى شئ خلاف الأرقام والحروف .

identifier 1 صحيح

identifier 2 صحيح بالرغم من أنه قد لايمكن تمييزه عن identifier 1 ، نظرا لأن طوله يزيد عن 8 خانات والثمانى خانات الأولى هى نفسها فى كل من الإسمين .

1 pointer ممنوع أن يبدأ الاسم برقم .

first record ممنوع وجود فراغ داخل الاسم .

to TO هي كلمة محجوزة .

total صحيح .

(٤٣) حدد : أي الأعداد التالية صحيح . وإذا كان خاطئا ؛ حدد سبب الخطأ .

-666 صحيح وهو عدد صحيح .

-666. خطأ – يجب أن يظهر رقم على كل جانب من جانبي العلامة العشرية .

-666.0 صحيح وهو عدد حقيقي .

-6,66e2 صحيح وهو عدد حقيقي .

483,500 خطأ – ممنوع استخدام الفواصل .

6- 166667e خطأ – ممنوع ظهور الفراغ .

(٤٤) حدد : أي من السلاسل التالية صحيح :

'Visit beautiful, sunny pittsburgh!' صحيح .

'The price is $56.50' صحيح .

'It's terribly cold out here!' خطأ لوجود علامة تنصيص واحدة داخل السلسلة .

(٤٥) حدد كيف يمكن للمعرفات التالية أن تصاحبها القيم الثابتة المناظرة لكل منها .

| المعرف | قيمة الثابت |
|---|---|
| count | 3 |
| offset | −2 |
| fraction | 0.333333 |
| color | blue |

```
CONST count = 3;
      offset = -2;
      fraction = 0.333333;
      color = 'blue';
```

ملاحظة : لاحظ أن هذه هي تعريفات definitions لثوابت .

(٤٦) حدد كيف يمكن للمعرفات التالية أن تصاحبها أنواع البيانات المناظرة لها .

| نوع البيانات | المعرف |
|---|---|
| صحيح | index |
| حقيقى | cmax |
| حقيقى | cmin |
| حرفى | code |
| بوليان | status |

```
VAR index : integer;
    cmax,cmin : real;
    code : char;
    status : boolean;
```

ملاحظة : لاحظ أن هذه هى توضيحات declarations للمتغيرات .

(٤٧) عرف نوع كل من التعبيرات التالية :

| | التعبير | النوع |
|---|---|---|
| (*a*) | `2*x+7` | عددى |
| (*b*) | `count >= 100` | بوليان |
| (*c*) | `sqr(value+3)/5` | عددى |
| (*d*) | `value = 666` | بوليان |
| (*e*) | `test` | عددى أو بوليان طبقا لنوع البيانات المصاحبة للاختبار |

(٤٨) فيما يلى عدة عبارات بسكال . عرف أى منها بسيط ، وأى منها مرتب .

(*a*) `area := length*width;` — بسيطة ( عبارة تحديد )

(*b*) `IF count = 100 THEN write(a,b,c);` — مرتبة ( شرطية )

(*c*) `GOTO 200;` — بسيطة ( تحويل غير شرطى )

(*d*) `read(length,width);` — بسيطة ( اتصال بإجراء )

(*e*) — مرتبة ( مركبة )

```
BEGIN
   read(length,width);
   area := length*width;
   write(length,width,area)
END
```

(*f*) `FOR index := 100 DOWNTO 1 DO write(index);` — مرتبة ( تكرارية )

(*g*) `mean := sqrt(sqr(a)+sqr(b)+sqr(c));` — بسيطة ( تحديد )

(٤٩) أى عبارة من عبارات السؤال السابق تتصل بإجراء ؟ وأى منها يحتوى على إشارات لدالة ؟

يمكن الاتصال بإجراءات باستخدام العبارات b , d , e , f . ويشار إلى دالة ( أى يتم الاتصال بها ) بآخر عبارة فقط من عبارات السؤال السابق .

## Supplementary Problems مشاكل متكاملة

(٥٠) حدد أى من المعرفات التالية صحيح .

(*a*) record1
(*b*) file2
(*c*) file
(*d*) name
(*e*) name and address
(*f*) employee_number
(*g*) 123-45-6789

(٥١) حدد أى من الأعداد التالية صحيح . وإذا ماكان العدد صحيحًا ؛ حدد إذا ماكان حقيقيًا ، أم صحيحًا .

(*a*) 0.5
(*b*) 27,822
(*c*) +93e12
(*d*) -4.083e-67
(*e*) 1.
(*f*) 40-55
(*g*) 12E12
(*h*) 131072
(*i*) 1.31072e5

(٥٢) حدد : أى من السلاسل التالية صحيح .

(*a*) '8:15 P.M.'
(*b*) "red, white and blue"
(*c*) 'Name:
(*d*) 'Chapter 3 (Cont''d)'
(*e*) '1.30172e5'
(*f*) 'NEW YORK, NY 10020'

(٥٣) حدد : كيف يمكن للمعرفات التالية أن تصاحبها القيم الثابتة المناظرة لكل منها .

| Identifier | Constant Value |
|---|---|
| month | july |
| fica | 123-45-6789 |
| price | $95.00 |
| gross | 2500.00 |
| partno | 48837 |
| bound | 0.00391 |

(٥٤) اكتب التوضيح الذى يصاحب كل من المعرفات التالية ، والبيانات المصاحبة لكل منها .

| Identifier | Data Type |
|---|---|
| period | char |
| terminal | boolean |
| status | char |
| index | integer |
| row | integer |
| clearance | real |

(٥٥) حدد نوع كل من التعبيرات التالية . عرف أي تعبير مكتوب بطريقة خاطئة .

(*a*) `counter := 87`
(*b*) `value`
(*c*) `sqr(first+second+third)`
(*d*) `cost <= maximum`
(*e*) `2*-x+y`
(*f*) `factor1*(sum1+sum2)/factor2`
(*g*) `color = 'blue'`

(٥٦) حدد أي من العبارات التالية بسيط ، وأيها مركب . عرف نوع كل عبارة .

(*a*) `net := gross-(fedtax+statetax+citytax);`

(*b*)
```
BEGIN
   tax := fedtax+statetax+citytax;
   net := gross-tax
END
```

(*c*) `FOR counter := start TO finish DO write(counter);`

(*d*) `IF counter < finish THEN counter := counter+1;`

(*e*) `root := sqrt(a+b+c+d);`

(*f*) `write('root=',root);`

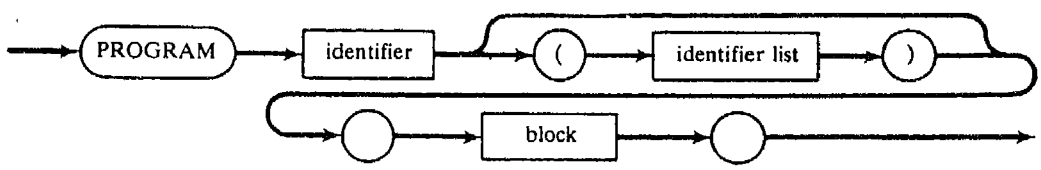
(*g*) `GOTO 17;`

(*h*) `new := new+old;`

(٥٧) أي عبارة من عبارات السؤال السابق تتصل بإجراءات ؟ وأيها تشير إلى دوال ؟

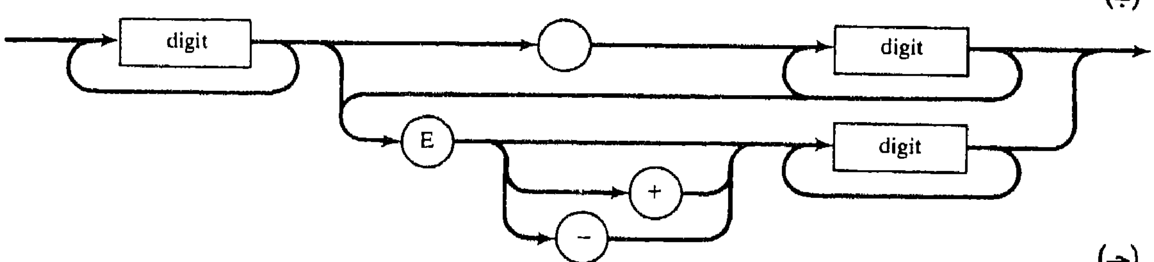
(٥٨) وضح معنى كل رسم من الرسومات التكوينية للبسكال التالية :

(أ)



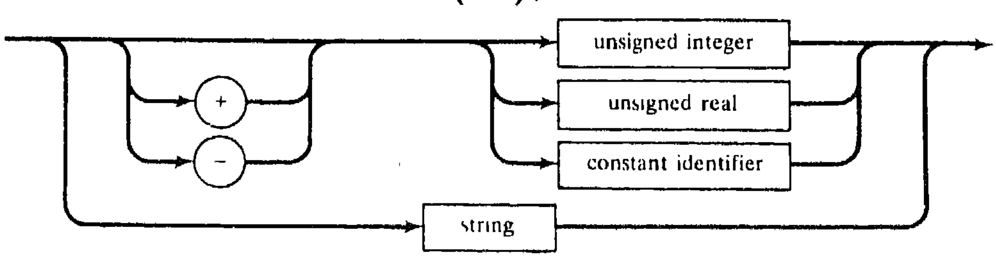

الشكل رقم (٢-٢)

(ب)




الشكل رقم (٢-٣)

(جـ)




الشكل رقم (٢-٤)

# الفصل الثالث

# بيانات من النوع البسيط

# Simple-Type Data

لقد أصبح القارئ معتادا الآن على بعض المفاهيم الأساسية للبسكال ، مما يجعلنا قادرين على الخوض فى تفاصيل أكبر لبيانات بسيطة النوع . وسوف نأخذ فى الاعتبار بصفة خاصة استخدامات البيانات البسيطة القياسية ، والتى تعتبر جزءا من لغة البسكال . وهذا النوع من البيانات يشمل : البيانات الصحيحة ، والحقيقية ، والحرفية ، وبيانات بوليان . كما سنناقش بالتفصيل العمليات والدوال القياسية التى يمكن استخدامها مع كل نوع من هذه الأنواع القياسية للبيانات ، كما يحتوى الفصل على معلومات إضافية عن تعبيرات واستخدامها فى عبارات التحديد .

وعادة مايشار إلى ثلاثة أنواع من البيانات القياسية ، وهى : البيانات الصحيحة ، والحرفية ، وبيانات بوليان بأنها بيانات ترتيبية ordinal ، حيث إن عناصر البيانات التى يتكون منها كل نوع من هذه الأنواع للبيانات تكون أعضاء فى فئة محددة ومرتبة . والبيانات البسيطة التى يقوم بتعريفها المستفيد ( بيانات متعددة enumerated وبيانات المدى الجزئى subrange ) تقع فى هذه المجموعة أيضا . لاحظ – على أية حال – أن البيانات من النوع الحقيقى لايمكن تقسيمها بهذه الطريقة . وعلى هذا ... فلاتقع فى هذه المجموعة .

## ١ – بيانات من النوع الصحيح : 1. INTEGER-TYPE DATA

البيانات من النوع الصحيح عبارة عن كميات من أرقام صحيحة . وتقع فى هذه المجموعة ثوابت ومتغيرات ودوال وتعبيرات من النوع الصحيح . وسبق أن ناقشنا قواعد كتابة الأرقام الصحيحة ، وقواعد تعريف وتوضيح الثوابت والمتغيرات الصحيحة . دعنا نركز الآن على المؤثرات التى يمكن استخدامها مع البيانات الصحيحة ، ثم نأخذ فى الاعتبار بعد ذلك قواعد كتابة التعبيرات الصحيحة .

وجميع المؤثرات المستخدمة لإجراء عمليات من النوع العددى تسمى مؤثرات حسابية arthmatic operators . وهناك ستة مؤثرات حسابية يمكن استخدامها مع عناصر صحيحة يمكن التأثير عليها . وخمسة من هذه المؤثرات تنتج عنها نتائج صحيحة ( أى نتيجة resultant من النوع الصحيح ) . أما المؤثر السادس ، فينتج عنه كمية من النوع الحقيقى . وهذه المؤثرات ملخصة على النحو التالى :

| المؤثر الحسابى | الغرض منه | نوع العناصر التى يمكن أن يؤثر عليها | نوع النتيجة |
|---|---|---|---|
| + | جمع | صحيح | صحيح |
| − | طرح | صحيح | صحيح |
| * | ضرب | صحيح | صحيح |
| / | قسمة | صحيح | حقيقى |
| DIV | قسمة مع حذف الكسر العشرى | صحيح | صحيح |
| MOD | قسمة ينتج عنها الرقم الصحيح الممثل للكسر العشرى المتبقى من عملية القسمة | صحيح | صحيح |

لاحظ أن مؤثر القسمة (/) هو الذى تنتج عنه كمية حقيقية ، بالرغم من أن العناصر التى يؤثر عليها صحيحة النوع . ولاحظ أيضا أنه لايوجد مؤثر أسى ... فلايستخدم مؤثر أسى فى البسكال .

## مـثال (٣-١)

افرض أن a و b متغيران صحيحان ، حددت لهما القيمتان 13 و 5 على التوالى . وفيما يلى عدة تعبيرات صحيحة والناتج منها .

| Expression | Value |
|---|---|
| a+b | 18 |
| a-b | 8 |
| a*b | 65 |
| a DIV b | 2 |
| a MOD b | 3 |
| a/b | 2.6 |

وهناك قواعد معينة تجب مراعاتها عند استخدام هذه المؤثرات فى تكوين تعبيرات عددية . وبعض هذه القواعد الأكثر شيوعا – والتى تتعلق بآخر عمليتين – قد أعطيا فى المثال ( وهما a MOD b و a/b ) . وسوف تناقش قواعد أخرى فيما بعد فى هذا الفصل .

(١) تكون المحصلة موجبة إذا كان لكل من العنصرين نفس الإشارة ، وإلا فستكون النتيجة سالبة .

(٢) مؤثرا القسمة ( / و DIV ) ، وكذلك المؤثر لباقى خارج القسمة MOD تتطلب ألا يكون العنصر الثانى صفرا .

(٣) استخدام المؤثر DIV مع عنصر سالب ينتج عنه حذف للكسر العشرى كلية ، أى أن النتيجة تصبح أقل من قيمتها الحقيقية .

(٤) طبقا للصيغة القياسية من ISO ، لايمكن أن يكون العنصر الثانى سالبا عند استخدام المؤثر MOD . وعلى أية حال تسمح عدة مترجمات بسكال بأن يكون مثل هذا العنصر سالبا . وفى مثل هذه الحالات تتحدد اشارة النتيجة ،

بحيث يتحقق الشرط التالى دائما ، بغض النظر عن إشارات العناصر الفردية .

## مـثال (٣-٢)

افرض أن i و j متغيران صحيحان ، وحددت لهما القيمتان 11 و 3- على التوالى . وفيما يلى عدة تعبيرات صحيحة والقيم المناظرة لها .

| Expression | Value |
|---|---|
| i+j | 8 |
| i-j | 14 |
| i*j | −33 |
| i DIV j | −3 |
| i MOD j | 2 (nonstandard) |
| i/j | −3.6666667 |

فاذا ماكانت قيمة i هى 11- وقيمة j هى 3 ، فإننا نحصل على مايلى :

| Expression | Value |
|---|---|
| i DIV j | −3 |
| i MOD j | −2 |
| i/j | −3.6666667 |

وأخيرا ... إذا كانت قيمة i هى 11- وقيمة j هى 3- ، فإننا نحصل على مايلى :

| Expression | Value |
|---|---|
| i DIV j | 3 |
| i MOD j | −2 (nonstandard) |
| i/j | 3.6666667 |

لاحظ أن الشرط ( غير القياسى ) التالى :

```
i = (i DIV j)*j + (i MOD j)
```

قد تحقق فى كل من المواقف سابقة الذكر .

## ٢ – بيانات من النوع الحقيقى : 2. REAL-TYPE DATA

تشير البيانات من النوع الحقيقى إلى عناصر بيانات تمثل كميات عددية حقيقية ، وتشمل هذه الثوابت والمتغيرات والدوال والتعبيرات الحقيقية .

وهناك أربعة مؤثرات حسابية يمكن استخدامها مع عناصر من النوع الحقيقى ، وهى :

| المؤثر الحسابى | الغرض منه | نوع العناصر | نوع النتيجة |
|---|---|---|---|
| + | جمع | حقيقى | حقيقى |
| − | طرح | حقيقى | حقيقى |
| * | ضرب | حقيقى | حقيقى |
| / | قسمة | حقيقى | حقيقى |

لاحظ أنه يمكن استخدام هذه المؤثرات مع عناصر من النوع الصحيح أيضا ، إلا أن المؤثرين DIV و MOD المستخدمين مع العناصر الصحيحة لايمكن استخدامها مع البيانات الحقيقية . ونذكر هنا مرة أخرى أنه لايوجد مؤثر أسى فى لغة البسكال .

مثال (٣ – ٣)

افرض أن v1 و v2 متغيران حقيقيان ، وأنه محدد لهما القيمتان 12.5 و 0.5 على التوالى . فيما يلى بعض التعبيرات الحقيقية البسيطة والنتيجة لكل منها .

| Expression | Value |
|---|---|
| v1+v2 | 13.0 |
| v1-v2 | 12.0 |
| v1*v2 | 6.25 |
| v1/v2 | 25.0 |

وهناك قواعد معينة يجب أن تتبع عند تكوين التعبيرات الحسابية بالبيانات الحقيقية . وفيما يلى ملخصا لثلاثة قواعد شائعة الاستخدام فى تكوين تعبيرات تحتوى على عنصرين اثنين فقط . وسوف نقدم قواعد أخرى فيما بعد فى هذا الفصل .

(١) النتيجة تكون موجبة إذا كان لكل من العنصرين نفس الإشارة . وإلا فإن إشارة النتيجة تكون سالبة .

(٢) يتطلب مؤثر القسمة (/) ألا يكون العنصر الثانى صفرا .

(٣) إذا كان أحد العناصر صحيحا ، والآخر حقيقيا ؛ فإن النتيجة تكون حقيقية دائما .

مـثال (٣-٤)

إذا كان r1 و r2 متغيرين حقيقيين لهما القيمتان 0.66- و 4.50 على التوالى . ففيما يلى عدة تعبيرات حقيقية والنتائج المناظرة لها .

| Expression | Value |
|---|---|
| r1+r2 | 3.84 |
| r1-r2 | −5.16 |
| r1*r2 | −2.97 |
| r1/r2 | −0.1466667 |

مـثال (٣-٥)

افرض أن i متغير من النوع الصحيح ، ومحدد له القيمة 2- ، وأن r متغير من النوع الحقيقى ، ومحدد له القيمة 1.2

التعبير 3*i*r

يمثل قيمة حقيقية تساوى 7.2- ، حيث 3x (-2) x 1.2 = 7.2.

## ٣ - بيانات من النوع الحرفى : 3. CHAR-TYPE DATA

البيانات من النوع الحرفى عبارة عن سلاسل مكونة من حرف واحد ، أو رقم واحد ، أو رمز خاص واحد ، على أن يكون موضوعا بين علامتى تنصيص . ويحتوى هذا النوع من البيانات على ثوابت ومعرفات تمثل ثوابت مكونة من حرف واحد ، أو رقم واحد ، أو رمز خاص واحد ، ومتغيرات من النوع الحرفى وبعض الدوال من النوع الحرفى .

ومجموعة الرموز التى يمكن استخدامها مع البيانات الحرفية تتغير من مترجم لمترجم آخر . وبصفة عامة ... فإن الحروف من A إلى Z ، سواء الكبيرة أم الصغيرة ، والأرقام من 0 إلى 9 ، والرموز الخاصة الشائعة الاستخدام كلها مقبولة .

مـثال (٣-٦)

فيما يلى عناصر بيانات حرفية النوع صحيحة

'p'   '5'   't'   '*'   ' '   ''''

لاحظ أن آخر عنصر يمثل علامة تنصيص واحدة كما سبقت الاشارة لذلك .

ومعظم أجهزة الكمبيوتر ، وكل أجهزة الميكروكمبيوتر تستخدم مجموعة رموز الشفرة الأمريكية القياسية لتبادل المعلومات American Standard Codes for Information Exchange (ASCII) ، والتى يحدث فيها تمثيل عددى من 7 بت لكل رمز ( وعلى هذا يكون هناك عدد يساوى $2^7 = 128$ من الرموز المختلفة ) . وتكون الرموز مرتبة طبقا للشفرة الخاصة بكل منها . والأرقام بصفة خاصة تكون مرتبة طبقا لتسلسلها العددى المناسب ، أى من 0 إلى 9 ، كما أن الحروف الأبجدية تكون مرتبة طبقا لترتيبها الأبجدى ، أى من A إلى Z . وهذا يسمح بمقارنة عناصر البيانات الحرفية التى تمثل رموز الشفرة الأمريكية القياسية لتبادل المعلومات مع بعضها طبقا لترتيبها النسبى المحدد لها فى هذه المجموعة .

ويحتوى جدول ٣ – ١ على جزء من مجموعة رموز الشفرة الأمريكية القياسية لتبادل المعلومات ، موضحا المكافئ العشرى لـ 7 بت الذى يمثل كل رمز من رموز المجموعة . لاحظ أن الأرقام تسبق الحروف ، وأن الحروف الكبيرة تسبق الحروف الصغيرة . لاحظ أيضا أنه هناك مجموعات صغيرة من الرموز الخاصة ، تفصل الأرقام والحروف الكبيرة والحروف الصغيرة ، أى تقع بين كل مجموعة من هذه المجموعات .

جدول (٣-١) جزء من مجموعة رموز ASCII

| ASCII Value | Character | ASCII Value | Character | ASCII Value | Character |
|---|---|---|---|---|---|
| 032 | blank | 063 | ? | 093 | ] |
| 033 | ! | 064 | @ | 094 | ^ |
| 034 | ” | 065 | A | 095 | — |
| 035 | # | 066 | B | 096 | |
| 036 | $ | 067 | C | 097 | a |
| 037 | % | 068 | D | 098 | b |
| 038 | & | 069 | E | 099 | c |
| 039 | ' | 070 | F | 100 | d |
| 040 | ( | 071 | G | 101 | e |
| 041 | ) | 072 | H | 102 | f |
| 042 | * | 073 | I | 103 | g |
| 043 | + | 074 | J | 104 | h |
| 044 | , | 075 | K | 105 | i |
| 045 | – | 076 | L | 106 | j |
| 046 | . | 077 | M | 107 | k |
| 047 | / | 078 | N | 108 | l |
| 048 | 0 | 079 | O | 109 | m |
| 049 | 1 | 080 | P | 110 | n |
| 050 | 2 | 081 | Q | 111 | o |
| 051 | 3 | 082 | R | 112 | p |
| 052 | 4 | 083 | S | 113 | q |
| 053 | 5 | 084 | T | 114 | r |
| 054 | 6 | 085 | U | 115 | s |
| 055 | 7 | 086 | V | 116 | t |
| 056 | 8 | 087 | W | 117 | u |
| 057 | 9 | 088 | X | 118 | v |
| 058 | : | 089 | Y | 119 | w |
| 059 | ; | 090 | Z | 120 | x |
| 060 | < | 091 | [ | 121 | y |
| 061 | = | 092 | \ | 122 | z |
| 062 | > | | | | |

## مثال (٣-٧)

يوضح جدول 3-1 أن الحرف A له الشفرة العشرية 65 فى الشفرة الأمريكية القياسية للتبادل العشرى ، وأن الحرف B له الشفرة العشرية 66 . وحيث إن 65 أقل من 66 ؛ فيعتبر A أقل من B . وعلى هذا ... فإن A يسبق B فى الترتيب . وبالمثل فإن A يسبق a ، لأن 65 أقل من 97 ، كما أن 0 يسبق 1 ، لأن 48 أقل من 49 .

ويوجد استثناء هام فى أجهزة الكمبيوتر IBM الكبيرة ، والتى تستخدم الشفرة الثنائية الموسعة للتبادل العشرى Extended Binary Coded Decimal Interchange Code (EBCDIC) فى ترتيب الرموز . وهذا هو مخطط مكون من 8 بت لكل رمز ، ويشتمل على 256 رمزا ($256 = 2^8$) . ومجموعة رموز الشفرة الثنائية الموسعة للتبادل العشرى هى صيغة مستقلة ، وهى ليست متوافقة مع الشفرة الأمريكية القياسية للتبادل العشرى . ويوضح الجدول ٣ - ٢ جزءا من شفرة الرموز فى الشفرة الثنائية الموسعة للتبادل العشرى . لاحظ أن الحروف الصغيرة تسبق الحروف الكبيرة فى هذه الشفرة ، كما أن الحروف الكبيرة تسبق الأرقام ، كما أنه هناك فجوات ( عدم استمرارية ) أيضا فى تسلسل شفرة الأرقام . ولاتزال الحروف والأرقام مرتبة طبقا لتسلسلها الطبيعى ، بحيث يمكن على أية حال مقارنة رموز الشفرة الثنائية الموسعة للتبادل العشرى طبقا لترتيبها النسبى داخل مجموعة الرموز .

جدول (٣-٢) جزء من مجموعة رموز EBCDIC

| EBCDIC Value | Character | EBCDIC Value | Character | EBCDIC Value | Character |
|---|---|---|---|---|---|
| 064 | blank | 132 | d | 200 | H |
| 074 | ] | 133 | e | 201 | I |
| 075 | . | 134 | f | 209 | J |
| 076 | < | 135 | g | 210 | K |
| 077 | ( | 136 | h | 211 | L |
| 078 | + | 137 | i | 212 | M |
| 079 | ! | 145 | j | 213 | N |
| 080 | & | 146 | k | 214 | O |
| 090 | [ | 147 | l | 215 | P |
| 091 | $ | 148 | m | 216 | Q |
| 092 | * | 149 | n | 217 | R |
| 093 | ) | 150 | o | 226 | S |
| 094 | ; | 151 | p | 227 | T |
| 095 | ^ | 152 | q | 228 | U |
| 096 | - | 153 | r | 229 | V |
| 097 | / | 162 | s | 230 | W |
| 108 | , | 163 | t | 231 | X |
| 109 | % | 164 | u | 232 | Y |
| 110 | — | 165 | v | 233 | Z |
| 111 | > | 166 | w | 240 | 0 |
| 112 | ? | 167 | x | 241 | 1 |
| 122 | : | 168 | y | 242 | 2 |
| 123 | # | 169 | z | 243 | 3 |
| 124 | @ | 193 | A | 244 | 4 |
| 125 |  | 194 | B | 245 | 5 |
| 126 | = | 195 | C | 246 | 6 |
| 127 | " | 196 | D | 247 | 7 |
| 129 | a | 197 | E | 248 | 8 |
| 130 | b | 198 | F | 249 | 9 |
| 131 | c | 199 | G |  |  |

### مثال (٣-٨)

يوضح جدول ٣ - ٢ أن الشفرة العشرية للحرف A هي 193 في الشفرة الثنائية الموسعة للتبادل العشرى ، وأن شفرة الحرف B هي 194 . وحيث إن 193 أقل من 194 ، فإن A يسبق B . وبالمثل فإن a تسبق A ، لأن 129 أقل من 193 ، كما أن 0 يسبق 1 ، لأن 240 أقل من 241

ويجب أن يحدد القارئ أى مجموعة الرموز المستخدمة فى الكمبيوتر المتاح له . كما يجب أن يكون مفهوما أن الكميات الصحيحة والأرقام من النوع الحرفى تمثل مع بعضها بطريقة مختلفة داخل الكمبيوتر . ويجب أن يكون المبرمج المبتدئ حريصا فى ألا يخلط بين هذه الأنواع من البيانات .

ولايمكن استخدام أى من المؤثرات الحسابية مع البيانات الحرفية ، حيث إن عناصر البيانات الحرفية لاتمثل كميات عددية . ويمكن على أية حال مقارنة البيانات الحرفية باستخدام المؤثرات العلاقية المذكورة فى القسم التالى .

## ٤ - بيانات من نوع بوليان : 4. BOOLEAN-TYPE DATA

البيانات من نوع بوليان تكون قيمتها إما صحيح أو خطأ . وتشمل هذه الفئة ثوابت ومتغيرات ودوال وتعبيرات من نوع بوليان . وتمثل القيمتان ( صحيح أو خطأ ) المستخدمتان مع بيانات بوليان فئة مرتبة تسبق خطأ فيها صحيح . ( لاحظ أن شفرة خطأ false هى 0 ، وشفرة صحيح true هى 1 ) .

وتتكون تعبيرات بوليان من مجموعة من المؤثرات من نفس نوع المؤثرات العلاقية relational operators . وتمثل هذه المؤثرات شروطا مختلفة للتساوى أو عدم التساوى . ويوجد سبعة مؤثرات علاقية فى البسكال ، وسوف نأخذ فى الاعتبار الستة مؤثرات التالية منها فقط فى هذا الفصل :

| مؤثر العلاقات | معناه |
|---|---|
| = | يساوى |
| <> | لايساوى |
| < | أقل من |
| <= | أقل من أو يساوى |
| > | أكبر من |
| >= | أكبر من أو يساوى |

وسوف يناقش المؤثر السابع IN فى الفصل الثانى عشر من الكتاب

ويمكن استخدام هذه المؤثرات الستة مع عناصر من أى نوع آخر غير نوع البوليان . وعند استخدامها مع عناصر غير عددية يشير عدم التساوى إلى ترتيب هذه العناصر طبقا للشفرة المستخدمة .

كما تستخدم بعض هذه المؤثرات ( وهى = و <> و => ، =< ) فى المقارنات ، كما هو مذكور فى الفصل الثانى عشر من الكتاب .

## مــثال (٣-٩)

فيما يلى بعض تعبيرات بوليان بسيطة تحتوى على عناصر عددية ( افرض أن كل من العنصرين داخل كل تعبير من نفس النوع ) .

| Exprèssion | Value |
|---|---|
| 2 = 3 | false |
| 2 < 3 | true |
| 0.6 >= 1.5 | false |
| 0.6 >= -1.5 | true |
| -4 <> 4 | true |
| 1.7 <= -2.2 | false |

## مــثال (٣-١٠)

افرض أن i و j متغيران من النوع الصحيح ، ومحدد لهما القيمتان 3 و 5- على التوالى . فيما يلى عدة تعبيرات بوليان تستخدم هذين المتغيرين .

| Expression | Value |
|---|---|
| i <= 10 | true |
| i+j > 0 | false |
| (i-j) < (i+j) | false |
| i-3 = j+5 | true |
| 2*i >= i DIV 2 | true |
| (i DIV 2) > (j+6) | false |

لاحظ أن العناصر يمكن أن تكون ثوابت أو متغيرات أو تعبيرات .

## مــثال (٣-١١)

افرض أن ch1 و ch2 هما متغيران حرفيان ، ومحدد لهما الحرفان P و T على التوالى ، وفيما يلى عدة تعبيرات بوليان تستخدم هذين المتغيرين .

| Expression | Value |
|---|---|
| ch1 = ch2 | false |
| ch2 = 'T' | true |
| ch1 = 'p' | false |
| ch1 < ch2 | true (because P precedes T) |
| ch2 > 'A' | true (because T succeeds A) |
| 'W' <> ch1 | true |

كما تحتوى لغة البسكال على ثلاثة مؤثرات منطقية أيضا . ويسمح مؤثران منهما باستخدام عناصر من نوع بوليان فى تكوين تعبيرات بوليان . أما الثالث ، فيستخدم فى نفى negate ( أو عكس reverse ) قيمة العنصر من نوع بوليان .

والمؤثرات المنطقية هى :

| المــؤثــر | معــنــاه |
|---|---|
| OR | يكون التعبير صحيحا إذا كان أى من العنصرين صحيحا ، أو إذا كان كل من العنصرين صحيحا . |
| AND | يكون التعبير صحيحا إذا كان كل من العنصرين صحيحا فقط |
| NOT | يستخدم هذا المؤثر كسابقه prefix لنفى العنصر من النوع بوليان . |

ولتجنب الخطأ فى ترتيب تنفيذ العمليات المنطقية ، يجب أن توضع عناصر بوليان بين قوسين .

مــثال (٣-١٢)

افرض أن n متغير صحيح ، محدد له القيمة 10 ، وأن s متغير حرفى يمثل الحرف A . فيما يلى عدة تعبيرات بوليان تستخدم هذين المتغيرين .

| Expression | Value |
|---|---|
| `(n > 0) AND (n < 20)` | true |
| `(n > 0) AND (n < 5)` | false |
| `(n > 0) OR (n < 5)` | true |
| `(n < 0) OR (n > 20)` | false |
| `(n = 10) AND (s = 'A')` | true |
| `(n <> 5) OR (s >= 'A')` | true |

مــثال (٣-١٣)

افرض أن تعبير بوليان $j > 6$ صحيحا ، وعلى هذا ... فإن التعبير

```
NOT (j > 6)
```

يكون خاطئا . كما أن التعبير

```
NOT (j <= 6)
```

يكون صحيحا ، حيث إن ( j < = 6 ) تكون خاطئة .

مــثال (٣-١٤)

افرض أن ch متغير حرفى يمثل الحرف G . وعلى هذا ... فإن التعبير ch > 'A' يكون صحيحا ، كما أن التعبير `NOT (ch > 'A')` يكون خاطئا .

والتعبير`ch = 'A'` يكون خاطئا ، كما أن التعبير `NOT (ch = 'A')` يكون صحيحا .

وسوف نرى كيفية استخدام تعبيرات بوليان فى برامج البسكال عندما نصل إلى الفصل السادس من الكتاب .

## ٥ - الثوابت القياسية : 5. STANDARD CONSTANTS

يحتوى البسكال على ثلاثة معرفات قياسية تمثل ثوابت ، وهى : maxint و false و true . ويحدد أول هذه المعرفات maxint أكبر قيمة لأى كمية من النوع الصحيح . أما الثابتان الآخران ، وهما false و true ، فيمثلان قيمتين يمكن تحديدهما لعناصر بيانات من نوع بوليان . ( يجب أن يتذكر القارئ مرة أخرى أن false و true يمثلان فئة مرتبة يسبق false فيها true ) .

## ٦ - الدوال القياسية : 6. STANDARD FUNCTIONS

يحتوى البسكال أيضا على عدد من الدوال القياسية التى تستخدم مع بيانات بسيطة مختلفة . ( كما يشار أيضا إلى الدوال القياسية بأنها دوال داخلية intrinsic ، أو دوال مبنية داخليا built-in ) . وتقبل بعض هذه الدوال نوعًا واحدًا من المؤثرات وتعود بقيمة من نفس النوع ، بينما تقبل دوال أخرى مؤشرات من نوع واحد ، وتعود بقيمة من نوع مختلف . وفيما يلى ملخصا لبعض هذه الدوال . ( انظر ملحق D لمزيد عن هذه الدوال ) .

| الدالة | الغرض منها | نوع المؤشر (x) | نوع النتيجة |
|---|---|---|---|
| abs(x) | حساب القيمة المطلقة لمقدار x | صحيح أو حقيقى | مثل نوع x |
| arctan (x) | حساب الظل العكسى للزاوية x | صحيح أو حقيقى | حقيقى |
| chr (x) | تحديد الرمز الذى يمثله المتغير x | صحيح | حرفى |
| cos(x) | حساب جيب تمام الزاوية x ( x بالتقدير الدائرى ) | صحيح أو حقيقى | حقيقى |
| exp (x) | حساب $e^x$ حيث e - 2.7182818 ، وهى أساس النظام الطبيعى للوغاريتمات | صحيح أو حقيقى | حقيقى |
| In (x) | حساب اللوغاريتم الطبيعى لـ x ( x > 0 ) | صحيح أو حقيقى | حقيقى |
| odd(x) | تحديد ماإذا كان x زوجيًا أو فرديًا ( تعود بقيمة صحيح إذا كان x فرديًا ، وقيمة خاطئ إذا كان x غير ذلك ) | صحيح | بوليان |
| ord (x) | تحديد الرقم الصحيح ( العشرى ) المقابل لشفرة الرمز x | حرفى | صحيح |
| pred (x) | تحديد مايسبق x | حرفى أو صحيح أو بوليان | مثل نوع x |
| round (x) | تقريب قيمة x لأقرب عدد صحيح . | حقيقى | صحيح |
| sin (x) | حساب جيب الزاوية x ( x بالتقدير الدائرى ) | صحيح أو حقيقى | حقيقى |
| sqr (x) | حساب مربع x | صحيح أو حقيقى | حقيقى |
| sqrt (x) | حساب الجذر التربيعى لـ x ( x > 0 ) | صحيح أو حقيقى | حقيقى |
| succ (x) | حساب مايلى x | حرفى أو صحيح أو بوليان | مثل نوع x |
| trunc (x) | يحذف الكسر من x | حقيقى | صحيح |

وهناك دالتان قياسيتان إضافيتان ، هما coln و cof ، وتستخدمان مع ملفات البيانات . وسوف تذكر هاتان الدالتان فى الفصل الرابع من الكتاب .

والغرض من معظم الدوال التى سبق ذكرها يجب أن يكون واضحا . وعلى أية حال ... فهناك قلة من الدول لايكون الغرض منها واضحا . وفيما يلى أمثلة توضيحية لاستخدامات الدول .

**مـثال (٣-١٥)**

تحسب الدالة abs (x) القيمة المطلقة للعدد الذى يمثله المؤشر x . فإذا ماكان diff متغيرا حقيقيا ، وكان محددا له القيمة 0.003- فإن الدالة abs(diff) تعود بالقيمة 0.003 ( لاحظ أن diff هو مؤشر فى الدالة ) .

**مـثال (٣-١٦)**

تستخدم الدالتان chr و ord لتحديد العلاقة بين أى رمز من رموز بسكال والشفرة الرقمية المناظرة له . وعلى هذا إذا كان الكمبيوتر مستخدما لشفرة ASCII ، فإن

```
chr(65)='A'          ord('A')=65
chr(112)='p'         ord('p')=112
chr(53)='5'          ord('5')=53
```

وهكذا لاحظ أن :

```
ord('A')=ord(chr(65))=65
```

وكذلك

```
chr(65)=chr(ord('A'))='A'
```

وهكذا

**مـثال (٣-١٧)**

حيث إن البيانات من النوع الصحيح والحرفى والبوليان كلها تمثل فئات مرتبة ، فيمكننا أن نحدد مايسبق أو مايتبع أى عنصر بيانات داخل إحدى هذه الفئات ( أو فى أى فئة مرتبة يقوم بتعريفها المستفيد ) ، وذلك باستخدام الدالتين pred و succ .

وعلى هذا

| | | |
|---|---|---|
| ( بيانات صحيحة ) | succ(3)=4 | pred(3)=2 |
| ( بيانات حرفية ) | succ('e')='f' | pred('e')='d' |
| ( بيانات بوليان ) | succ(false)=true | pred(true)=false |

لاحظ أيضا أن فئة رموز ASCII تحقق شروطا قبل

```
pred('e')=chr(ord('e')-1)
```

وكذلك

```
succ('e')=chr(ord('e')+1)
```

ويجب أن يفهم أن الدالتين pred و succ لاتستخدمان مع بيانات من النوع الحقيقى .

### مثال (٣-١٨)

يمكن للدالتين round و trunc أن تقبلا أعدادا حقيقية سالبة أو موجبة . وتعامل الأعداد السالبة مثل الأعداد الموجبة ، مع إضافة إشارة السالب بعد التقريب أو حذف الكسر . وعلى ذلك ، فإن :

```
round(2.3)=2        trunc(2.3)=2
round(3.7)=4        trunc(3.7)=3
round(-1.8)=-2      trunc(-1.8)=-1
round(-6.1)=-6      trunc(-6.1)=-6
```

ويمكن استخدام العديد من هذه الدوال مع بيانات أخرى غير قياسية ، أى يتم تعريفها من قبل المستفيد أيضا . وسوف نذكر هذه التطبيقات فيما بعد فى هذا الكتاب كلما كانت هناك حاجة لذلك .

كما يجب أن يتذكر القارئ مرة أخرى أن المؤشرات الموجودة فى دليل الدالة يمكن أن تكون ثوابت أو متغيرات أو تعبيرات ، أو حتى مؤشرات لدوال أخرى . والقيد الوحيد هو أن هذه المؤشرات يجب أن تكون من النوع المناسب . وسوف نتعرض المزيد عن المؤشرات فى الفصل السابع من الكتاب ، حيث نتعرض لموضوع الإجراءات والدوال بتفصيلات أكثر .

## ٧ - المزيد عن التعبيرات : 7. MORE ABOUT EXPRESSIONS

يمكن أن يصبح التعبير معقدا فى بعض الأحيان بسبب وجود مؤثرات متعددة داخل التعبير . وفى مثل هذه الحالات يصبح من اللازم تحديد ترتيب تنفيذ العمليات المختلفة . ويمكن تحديد هذا الترتيب بواسطة أولوية المؤثر الطبيعى operator precedence ، والموجود فى لغة البسكال . ومجموعات الأولويات هى كما يلى :

| الأولوية | المؤثر |
|---|---|
| 1 ( الأعلى ) | NOT |
| 2 | * / DIV MOD AND |
| 3 | + - OR |
| 4 ( الأدنى ) | * <> < <= > >= IN |

وفى نفس مجموعة الأولويات تنفذ العمليات طبقا لظهورها ، أى الأولى فالتالية لها ، فالتالية لها ، من اليسار إلى اليمين .

### مثال (٣-١٩)

التعبير العددى التالى :

```
a-b/c*sqrt(d)
```

يناظر المعادلة الجبرية التالية

$$a - [(b/c) \times \sqrt{d}$$

فإذا كانت قيم المتغيرات d , c , b , a لها القيم 1 , 2 , 3 , 4 على التوالى . فسوف يمثل التعبير القيمة -0.33333333 ، حيث :

$$1 - [(2/3) \times \sqrt{4}] = 1 - (4/3) = -1/3$$

لاحظ أن عملية القسمة أجريت أولا ، حيث إنها تقع فى مجموعة لها أولوية مرتفعة عن أولوية الطرح . ونتيجة القسمة ضربت بعد ذلك فى 4 ( طبقا لقاعدة من اليسار إلى اليمين فى أولويات التنفيذ ) . وأخيرا طرح حاصل الضرب هذا من قيمة أول متغير .

ويمكن تغيير الأولويات الطبيعية ، وذلك باستخدام الأقواس ، وذلك لأداء العمليات التى يحتويها التعبير بأى أولويات مطلوبة . وفى مثل هذه الحالات تنفذ العمليات الموجودة بين القوسين الداخليين جدا ، تليها العمليات الموجودة بين القوسين التاليين ، وهكذا .

## مثال (٣-٢٠)

التعبير الحسابى التالى :

```
(a-b)/(c*sqrt(d))
```

يناظر المعادلة الجبرية التالية :

$$(a - b)/(c \times \sqrt{d})$$

فإذا كانت قيم المتغيرات d,c,b,a هى 4,3,2,1 على التوالى ، فإن التعبير الحسابى يمثل القيمة -0.16666667 ، حيث إن :

$$(1 - 2)/(3 \times \sqrt{4}) = -1/6 = -0.1666667$$

( قارن هذه النتيجة بنتيجة مثال ١٩ - ٣ ) .

## مثال (٣-٢١)

اعتبر تعبير بوليان التالى :

```
(x > 0) OR (y < 10)
```

حيث إن x و y متغيران صحيحان . ويكون هذا التعبير صحيحا إذا ماكانت قيمة x أكبر من الصفر أو إذا كانت قيمة y أقل من 10 ( أو إذا تحقق الشرطان معا ) . أما إذا لم يتحقق الشرطان ، فإن قيمة التعبير تكون خاطئة .

لاحظ أن الأقواس مطلوبة فى هذا التعبير . فبدون الأقواس يحاول البسكال أن يقوم التعبير التالى :

```
0 OR y
```

والذى لامعنى له ( نظرا لأنه لايمكن استخدام المؤثر المنطقى OR مع عناصر عددية ) .

مـثال (٣-٢٢)

اعتبر التعبير العددى التالى :

```
2*((a MOD 5)*(4+(b-3)/sqr(c+2)))
```

إذا كانت قيم المتغيرات a و b و c هـى 8 و 15 و 4- على التوالى ، فسوف يتم تقويم هذا التعبير على النحو التالى :

```
2*((3)*(4+12/sqr(-2))) = 2*(3*(4+12/4)) = 2*(3*(4+3)) = 42
```

وفى بعض الأحيان يكون استخدام الأقواس لتوضيح التعبير مفيدا ، وذلك بالرغم من أنه قد لايكون مطلوبا وجود الأقواس . ومن ناحية أخرى ... يجب تجنب استخدام التعبيرات المعقدة جدا كلما كان ذلك ممكنا ، حيث إن مثل هذه التعبيرات تمثل مصدرا متكررا للخطأ ( يحتوى آخر مثال على مثل هذه التعبيرات ) .

ويجب أن تتبع هذه القواعد ( بالإضافة إلى القواعد التى سبق ذكرها فى هذا الفصل ) عند كتابة تعبيرات عددية وتعبيرات بوليان

(١) لايمكن أن تظهر معرفات غير معرفة داخل التعبير . ( وفى كلمات أخرى ... يجب أن تحدد قيمة معينة لكل معرف قبل أن يمكن استخدامه داخل التعبير ) .

(٢) ظهور إشـارة سالب تسـبق أحد المعرفـات يكافـئ ضـرب المعرف فى 1- ، وعلـى هـذا ... فـإن a*b- يكافـئ -1*a*b .

(٣) ممنوع ظهور المؤثرات الحسابية وراء بعضها مباشرة . وعلى هذا ... فإن التعبير a*-b غير مسمـوح به . أما التعـبير a*(-b) فمسموح به .

(٤) لايمكن أن تكون العمليات الحسابية مشمولة . وعلى هذا ... فالتعبير 2(x+y) خاطئ . أما التعبير 2*(x+y) ، فهو صـحيح .

(٥) لايمكن أداء العمليات الحسابية على بيانات حرفية أو بيانات بوليان . وعلى هذا ... فإن تعبيرات مثل :

```
'A'+'B'
```

أو مثل :

```
(n > 0) + (n < 20)
```

غير مسموح بها .

(٦) يجب أن يكون هناك اتزان فى الأقواس ، بمعنى أن عدد الأقواس المفتوحة يجب أن يتساوى مع عدد الأقواس المغلقة .

## ٨ - عبارة التحديد : 8. THE ASSIGNMENT STATEMENT

سبق أن رأينا أن عبارة التحديد هى عبارة بسيطة تستخدم فى تحديد عناصر بيانات لمتغيرات . وتكتب هذه العبارة على النحو التالى :

```
variable := data item
```

ويمكن أن يكون عنصر البيانات فرديا ( مثل ثابت ، أو متغير آخر ، أو دليل دالة ) أو يكون تعبيرا . ويجب على أية حال أن يكون عنصر البيانات من نفس نوع المتغير المحدد له . ( وهناك استثناء واحد لهذه القاعدة ، وهو أنه يمكن تحديد عنصر بيانات صحيح لمتغير حقيقى ) .

## مثال (٣-٢٣)

فيما يلى عبارة تحديد تقليدية لمساحة الدائرة

```
area := 3.14159*sqr(radius);
```

وتسبب هذه العبارة تحديد قيمة للتعبير

```
3.14159*sqr(radius)
```

وتحدد هذه القيمة للمتغير area . وعلى هذا ... فإذا كانت قيمة نصف القطر radius هى10.0 ، فإن القيمة التى تحدد للمتغير المساحة area هى 314.159 .

والفاصلة المنقوطة الموجودة فى نهاية العبارة ليست فى واقع الأمر جزءا من العبارة نفسها ، ولكنها فاصل separator يوضح أن العبارات انتهت ، وأن مايلى هذه الفاصلة هو بداية عبارة جديدة . وتستخدم الفاصلة المنقوطة مع كل عبارات البسكال ، ولايقتصر استخدامها على أى نوع من العبارات ، مثل عبارات التحديد .

وعادة ماتتشابه عبارات التحديد العددية مع المعادلات الجبرية . وليس هذا صحيحا دائما على أية حال كما يظهر ذلك من المثال التالى .

## مثال (٣-٢٤)

اعتبر عبارة التحديد التالية :

```
count := count+1;
```

تتسبب هذه العبارة فى زيادة القيمة الحالية للمتغير count بمقدار 1. ( وهذا يسمى بالعداد ) . ومن ناحية الجبر ، فإن هذه العبارة لامعنى لها .

ويمكن تحقيق نفس الشئ بكتابة مايلى :

```
count := succ(count);
```

حيث succ هى دالة قياسية ، كما سبق ذكره .

ولاتحتاج عبارة التحديد أن تقتصر على البيانات العددية فقط كما هو موضح فى المثالين التاليين .

## مثال (٣-٢٥)

افرض أن state هو متغير حرفى . وعلى هذا ... فإن عبارة التحديد التالية :

```
state := 'S';
```

تحدد أن قيمة state هى الحرف S .

مـثال (٣-٢٦)

افرض أن flag هو متغير بوليان ، وأن x , y هما متغيران صحيحان . وعلى هذا ... فإن العبارة التالية :

```
flag := (x > 0) OR (y < 10);
```

تتسبب فى وضع قيمة صحيح أو خاطئ للمتغير flag طبقا لقيمة التعبير البولياني . وعلى هذا ... فإن قيمة flag تكون صحيح إذا كانت قيمة x أكبر من صفر ، أو إذا كانت قيمة y أقل من 10 ( أو إذا تحقق الشرطان ) وإلا فإن قيمة flag تكون خاطئ .

## أسئلة للمراجعة : Review Questions

(١) ماذا تمثل البيانات من النوع الصحيح ؟

(٢) ماهو المؤثر الحسابى ؟ أى المؤثرات الحسابية يمكن استخدامها مع البيانات الصحيحة ؟ وماهو الغرض من استخدام كل من هذه المؤثرات ؟

(٣) ماهو الفرق بين مؤثر القسمة (/) ومؤثر الحذف (DIV) عند استخدام عناصر بيانات صحيحة ؟

(٤) لخص قواعد استخدام أحد المؤثرات الحسابية مع عناصر بيانات صحيحة ، ولخص بصفة خاصة القيود الموجودة على استخدام مؤثرى القسمة ( / و DIV ) والمؤثر MOD .

(٥) ماذا تمثل البيانات من النوع الحقيقى ؟

(٦) ماهى المؤثرات الحسابية التى يمكن استخدامها مع البيانات الحقيقية ؟ وماهو الغرض من استخدام كل من هذه المؤثرات ؟

(٧) لخص قواعد استخدام أى مؤثر حسابى مع عنصرى بيانات حقيقيين .

(٨) مانوع النتيجة التى يتم الحصول عليها اذا ماستخدم احد المؤثرات الحسابية مع عنصر بيانات صحيح وعنصر بيانات حقيقى ؟

(٩) ماذا تمثل البيانات من النوع الحرفى ؟ وماهى الرموز التى يمكن استخدامها مع البيانات الحرفية ؟

(١٠) ماذا تعنى مجموعة رموز ASCII ؟ ومامدى استخدام هذه المجموعة ؟

(١١) ماهى الصورة العامة لترتيب الرموز فى مجموعة رموز ASCII ؟

(١٢) ماهى مجموعة رموز EBCDIC ؟ وفى أى أنواع أجهزة الكمبيوتر يمكن أن توجد هذه المجموعة ؟

(١٣) ماهى الصورة العامة لترتيب الرموز فى مجموعة رموز EBCDIC ؟

(١٤) مانوع العمليات الذى يمكن اداؤها على البيانات الحرفية ؟ وماهى المؤثرات المستخدمة ؟

(١٥) ماذا تمثل البيانات من نوع بوليان ؟

(١٦) كيف يتم ترتيب عناصر البيانات من نوع بوليان ؟

(١٧) ماهى المؤثرات العلاقية ؟ وماهو الغرض من كل منها ؟ ومع أى نوع من أنواع عناصر البيانات تستخدم هذه المؤثرات ؟ وماهى النتيجة التى يتم الحصول عليها فى كل حالة ؟

(١٨) ماهو تفسير مؤثر علاقى من نوع عدم التساوى ( مثل <> ) والمستخدم مع عناصر حرفية ؟

(١٩) ماهى المؤثرات المنطقية ؟ وماهو الغرض من كل منها ؟ ومع أى عناصر بيانات يمكن استخدامها ؟ وماهو نوع النتيجة التى يتم الحصول عليها ؟

(٢٠) ماهى الثلاثة ثوابت القياسية الموجودة فى البسكال ؟ ومانوع عنصر بيانات كل ثابت من هذه الثوابت ؟ وماالغرض من كل هذه الثوابت ؟

(٢١) ماهى الدوال القياسية ؟ وماهى الأسماء الأخرى لها ؟

(٢٢) ماهو الغرض من دالة abs ؟ ومع أى نوع من أنواع المؤشرات تستخدم هذه الدالة ؟ وماهو نوع النتيجة التى يتم الحصول عليها ؟

(٢٣) ماهو الغرض من دالتى chr و ord ؟ ومع أى نوع من أنواع المؤشرات تستخدم هاتان الدالتان ؟ وماهو نوع النتائج التى يتم الحصول عليها ؟

(٢٤) ماهو الغرض من دالتى pred و succ ؟ ومع أى نوع من أنواع المؤشرات تستخدم هاتان الدالتان ؟ وماهو نوع النتائج التى يتم الحصول عليها ؟

(٢٥) عند استخدام مجموعة رموز ASCII ، فماهى العلاقة الموجودة بين دالتى chr و ord ، ودالتى pred و succ ؟ ولماذا لاتكون هذه العلاقة صحيحة مع كل مجموعات الرموز ؟

(٢٦) ماهو الغرض من استخدام دالتى round و trunc ؟ ومع أى نوع من أنواع المؤشرات تستخدم هاتان الدالتان ؟ وماهو نوع النتائج التى يتم الحصول عليها ؟

(٢٧) كيف تعامل دالّتا round و trunc المؤشرات السالبة ؟

(٢٨) وضح ماهو معنى أولوية المؤثر operator precedence ، ولخص تكوينه .

(٢٩) ماهو ترتيب تنفيذ العمليات الموجودة داخل إحدى مجموعات الأولويات ؟

(٣٠) كيف يمكن تغيير الأولويات المعتادة للتنفيذ ؟

(٣١) ماهى أولويات تنفيذ العمليات الموجودة داخل تعبير يحتوى على عدة أقواس متداخلة ؟

(٣٢) تحت أى ظروف يجب أن توجد الأقواس داخل التعبير ؟ وتحت أى ظروف يجب تجنب وجود الأقواس داخل التعبير ؟

(٣٣) لخص القواعد التى تجب مراعاتها عند عمل تعبيرات عددية وتعبيرات بوليان .

(٣٤) ماهو الغرض من عبارة التحديد ؟ وماهى القيود الموجودة على نوع البيانات التى تحدد للمتغير ؟

(٣٥) ماهو الغرض من الفاصلة المنقوطة التى تظهر فى نهاية عبارة التحديد ؟ وهل الفاصلة المنقوطة جزء فعلا من عبارة التحديد ؟

(٣٦) هل هناك علاقة بين عبارة تحديد للنوع الصحيح والمعادلة الجبرية ؟

## مسائل محلولة : Solved Problems

(٣٧) افرض أن a , b , c متغيرات عددية محدد لها القيم التالية :

| المتغير | النوع | القيمة |
|---|---|---|
| a | حقيقى | 5.7 |
| b | حقيقى | 8.2 |
| c | صحيح | 7 |
| d | صحيح | 4 |

وتستخدم هذه المتغيرات فى التعبيرات العددية التالية . حدد فى كل حالة من هذه الحالات نوع التعبير ، والقيمة التى يمثلها .

| التعبير العددى | النوع | القيمة |
|---|---|---|
| sqr(a+b)/(c+d) | حقيقى | 17.5645 |
| 6*(c MOD d) | صحيح | 18 |
| 6.0*(c MOD d) | حقيقى | 18.0 |
| (c DIV d)+(c MOD d) | صحيح | 4 |
| (c MOD d)/2 | حقيقى | 1.5 |
| trunc(a-b) | صحيح | −2 |
| trunc(a-b) DIV c | صحيح | 0 |
| (c DIV d)/(-a) | حقيقى | −0.17543860 |

(٣٨) فيما يلى العديد من تعبيرات بوليان . حدد القيمة التى يمثلها كل منها . افرض أن المتغيرات a , b , c , d لها نفس قيم السؤال السابق ، وأن الحرف عبارة عن متغير حرفى محدد له الرمز w .

| تعبير بوليان | القيمة |
|---|---|
| a < b | صحيح |
| letter = '$' | خطأ |
| 'q' < 'r' | صحيح |
| abs(a-b) > 2.0 | صحيح |
| (c <= d) OR (letter = 'Z') | خطأ |
| NOT (5 = pred(6)) | خطأ |
| 100*(c+d) = maxint | خطأ |
| true | صحيح |
| (c > 0) AND (c <> 7) | خطأ |

(٣٩) فيما يلى عدة تعبيرات عددية وتعبيرات بوليان . بعضها خاطئ . حدد هذه التعبيرات الخاطئة ، وسبب الخطأ .

| التعبير | الصحة |
|---|---|
| `2*a*b/(c-1)-u/abs(3*(v-w))` | – صحيح . |
| `2*a*-b/(c-1)-u/abs(3*(v-w))` | – غير صحيح ( مؤثرين متتاليين ) |
| `'a'+'b'+'c'` | – غير صحيح ( لايمكن استخدام مؤثرات حسابية مع أنواع بيانات حرفية ) |
| `succ('E') = 'F'` | – صحيح . |
| `x > 0 AND x < 100` | – غير صحيح ( يجب أن توضع مؤثرات بوليان بين أقواس ) |
| `(a+b+c) <= (c+d+e)` | – صحيح . |
| `2*n+1 <> 'a'` | – غير صحيح ( العوامل غير متوفقة ) |
| `round(99.7) = 100` | – صحيح . |
| `((x-y)/3+(abs(u-v))` | – غير صحيح ( الأقواس غير متوازنة ) |
| `(-5(x-y)/3+abs(u-v))` | – غير صحيح ( علامة الضرب غير موجودة ) |

## مشاكل متكاملة : Supplementary Problems

(٤٠) افرض أن d , c , b , a هى متغيرات عددية محدد لها القيم التالية :

| Variable | Type | Value |
|---|---|---|
| a | integer | 8 |
| b | integer | 5 |
| c | real | 4.3 |
| d | real | 0.8 |
| e | real | −2.2 |

تستخدم هذه المتغيرات فى التعبيرات العددية التالية . حدد نوع كل تعبير من هذه التعبيرات ، والقيمة التى يمثلها .

(*a*) `(b-a)/sqr(d-e)`
(*b*) `round((c+d)/e)`
(*c*) `(a DIV b)/(a MOD b)`
(*d*) `(a-2*b)*trunc(3*c-d+2*e)`
(*e*) `0.01*(a-b)`
(*f*) `trunc(3*sqrt(abs(d+e)))`

(٤١) فيما يلى عدة تعبيرات بوليان . حدد القيمة التى يمثلها كل من هذه التعبيرات . افرض أن المتغيرات e , d , c , b , a لها نفس القيم الموجودة فى المسألة السابقة ، وأن n , y متغيرات حرفية محدد لها الحرفان N , Y على التوالى .

(*a*) `c < d+e`
(*b*) `(y = 'Y') AND (n = 'N')`
(*c*) `trunc(c+d) <= 10.0`
(*d*) `(a = 8) OR (b = 8)`
(*e*) `true OR false`
(*f*) `NOT (y < 'z')`
(*g*) `(a >= 100) AND (b <= maxint)`
(*h*) `odd(a-b)`

(٤٢) حدد قيمة كل تعبير من التعبيرات التالية افرض أن الرموز مأخوذة طبقا لنظام شفرة ASCII .

(*a*) abs(-4.667)
(*b*) chr(67)
(*c*) ord('e')
(*d*) pred(10)
(*e*) pred('e')
(*f*) pred(true)
(*g*) succ(10)
(*h*) succ(e)
(*i*) succ(false)
(*j*) odd(10)
(*k*) odd(15)
(*l*) round(2.2)
(*m*) round(2.8)
(*n*) round(-2.2)
(*o*) round(-2.8)
(*p*) trunc(2.2)
(*q*) trunc(2.8)
(*r*) trunc(-2.2)
(*s*) trunc(-2.8)
(*t*) chr(ord('8')+4)
(*u*) ord(pred('A'))
(*v*) chr(ord('g'))

(٤٣) أى إجابة من إجابات المسألة السابقة سوف تتغير إذا ما استخدمت مجموعة رموز EBCDIC ؟

(٤٤) فيما يلى عدة تعبيرات عددية وتعبيرات بوليان . بعض هذه التعبيرات خاطئ . حدد التعبيرات الخاطئة ، وسبب الخطأ .

(*a*) sum DIV 0.005
(*b*) 200*sum
(*c*) '100'+'27'+440'
(*d*) true = false
(*e*) (a-b)/(2g+7)
(*f*) 'D'=pred('E')
(*g*) value = 0.0 OR count < 100
(*h*) slope <> m*x+b
(*i*) sqrt(a+b) < succ('G')
(*j*) (a+abs(c1+c2)/((c1+c3)*(c2+c3))
(*k*) 2*maxint+3
(*l*) round(a*-b)
(*m*) odd(12.82)-6.6
(*n*) -b+sqrt(sqr(b)-4*a*c)/(2*a)

(٤٥) يحتوى أحد برامج البسكال المتغيرات التالية :

| Variable | Data Type |
|---|---|
| gross | real |
| net | real |
| tax | real |
| employee | integer |
| status | char |
| sex | char |
| exempt | boolean |

بين كيف يتم توضيح هذه المتغيرات فى البرنامج .

## الفصل الرابع

# إدخال وإخراج البيانات

# Data Input and Output

يهتم هذه الفصل بطرق قراءة البيانات داخل الكمبيوتر ، وكتابة الكمبيوتر للمخرجات . وسوف نرى أنه يمكن تحقيق ذلك بسهولة ، وذلك باستخدام عبارات خاصة للمدخلات والمخرجات موجودة فى لغة البسكال . وبعد أن نتعلم كيفية أداء عمليات المدخلات والمخرجات هذه ، فإننا نكون قادرين على كتابة العديد من برامج البسكال البسيطة .

## ١ – ملفات المدخلات والمخرجات : 1. INPUT AND OUTPUT FILES

دعنا نعتبر طريقة تحويل بيانات المدخلات إلى الكمبيوتر ، وتحويل بيانات المخرجات من الكمبيوتر . وسوف نعتبر بيئة ( غير متداخلة ) لجهاز كمبيوتر كبير .

يجب وضع عنصر بيانات المدخلات فى ملف منفصل ، يسمى ملف مدخلات input file قبل تشغيل البرنامج المناظر له . وفى معظم الأحوال يتم انتاج هذا الملف عن طريق منقح للنصوص ، كالمنقح المذكور فى الفصل الخامس من الكتاب . كما يمكن أيضا إدخاله فى الكمبيوتر عن طريق بطاقات مثقبة ، وذلك باستخدام آلة التثقيب وقارئ البطاقات . وفى أى حالة من هذه الحالات تخزن عناصر البيانات على التوالى بنفس الترتيب الذى تم إدخالها به . وسوف تجمع عناصر البيانات الفردية فى أسطر lines منطقية ، حيث يناظر كل من هذه الأسطر سطرا من البيانات التى تم إدخالها ، أو يناظر بطاقة مثقبة .

وتنقل بيانات المخرجات من ذاكرة الكمبيوتر إلى ملف مخرجات output file ، والذى يشبه ملف المدخلات ، لكنه عكسه وتجمع عناصر البيانات مرة أخرى على التوالى ، وبنفس الترتيب التى كتبت به . سوف تجمع عناصر البيانات مرة أخرى فى أسطر منطقية ، والتى تناظر الأسطر الطبيعية ( أو الواقعية ) للمخرجات عند طباعة البيانات أو عرضها على الشاشة .

ويجب أن يكون مفهوما أن ملف بيانات المدخلات وملف بيانات المخرجات يخزنان داخل الكمبيوتر ككيانين منفصلين ، فهما ليسا أجزاء من برنامج البسكال . وتصاحب هذه الملفات برنامج البسكال على أية حال بتشغيلها كمؤشرات داخل عنوان البرنامج .

### مـثال (٤ –١)

يحتوى برنامج بسكال على العنوان التالى للبرنامج :

```
PROGRAM payroll(input,output);
```

ويحدد هذا العنوان أن اسم البرنامج هو payroll ، وأن البرنامج يستخدم ملف مدخلات وملف مخرجات .

ولايحتاج البرنامج إلى أن يستخدم كل من ملفى مدخلات ومخرجات ، وذلك بالرغم من أن معظم البرامج تفعل ذلك ، حيث إن معظم البرامج تتطلب بيانات مدخلات ، وتنتج بيانات مخرجات بعد تشغيلها .

مـثال (٤ -٢)

يستخدم برنامج بسكال اسمه primes فى إنتاج أول أثنى عشر عددا أوليا . ويحتوى هذا البرنامج على العنوان التالى :

```
PROGRAM primes(output);
```

لاحظ أن هذا البرنامج يستخدم ملف مخرجات فقط ، حيث إنه لايحتاج إلى بيانات مدخلات . فإذا ماعدل البرنامج على أية حال ، بحيث إنه ينقح أول n من الأعداد الأولية ، حيث n إحدى القيم التى يتم إدخالها للبرنامج ، فيجب أن يتغير عنوان البرنامج ليأخذ الصورة التالية :

```
PROGRAM primes(input,output);
```

ومثل هذا التعديل يعتبر فكرة جيدة ، حيث إن البرنامج يصبح عاما ، ولايقتصر على عدد محدد من الأعداد الأولية .

ويجب أن يتذكر القارئ أنه هناك بعض صيغ متداخلة للغة البسكال تسمح بأداء عمليات المدخلات والمخرجات مباشرة أثناء تنفيذ البرنامج . وفى مثل هذه الحالات لاتكون هناك حاجة إلى ملفات المدخلات والمخرجات . وهذا شائع الاستخدام مع أجهزة الميكروكمبيوتر بصفة خاصة .

## ٢ - عبارة اقرأ : 2. THE READ STATEMENT

تستخدم عبارة اقرأ لقراءة عناصر بيانات من ملف المدخلات ، ولتحديد هذه العناصر لمتغيرات صحيحة أو حقيقية أو حرفية . وتكتب هذه العبارة على النحو التالى :

read(*input variables*)

وتفصل متغيرات المدخلات بواسطة فواصل . ( لاحظ أنه لايمكن أن تظهر متغيرات بوليان داخل قائمة متغيرات المدخلات ) .

مـثال (٤ -٣)

فيما يلى عبارة اقرأ نمطية

```
read(a,b,c);
```

تتسبب هذه العبارة فى قراءة ثلاثة عناصر بيانات من ملف مدخلات ، وتحدد هذه العناصر لثلاثة متغيـرات هـى : c , b , a على التوالى .

وتقرأ عناصر البيانات من ملف المدخلات وتحدد للمتغيرات المناظرة لها بنفس الترتيب الذى تخزن به . ويجب أن يكون كل متغير من نفس نوع عنصر البيانات المناظر له . ( استثناء : يمكن تحديد عدد صحيح لمتغير حقيقى ) . ويمكن قراءة كل عنصر بيانات مرة واحدة فقط .

مـثال (٤ - ٤)

فيما يلى جزء من برنامج بسكال

(انظر الصفحة التالية)

```
VAR a,b : real;
    i,j : integer;
    p,q : char;
     .
     .
     .
read(a,b,i,j,p,q);
```

تتسبب عبارة اقرأ فى قراءة عددين حقيقيين ، وعددين صحيحين ، ورمزين فرديين من ملف المدخلات مع تحديدها للمتغيرات q , p , j , i , b , a على التوالى .

يجب الحذر عند وضع عناصر بيانات المدخلات . فيجب فصل عناصر البيانات العددية عن بعضها بفراغات أو بمؤشر انتهاء السطر . وحتى نكون أكثر دقة ، يجب أن يسبق عنصر البيانات العددى فراغ واحد أو أكثر ، أو مؤشر واحد لانتهاء السطر أو أكثر . ويمكن كتابة الأعداد الحقيقية باستخدام أس ، أو بدون استخدامه . والأكثر من هذا ... يمكن كتابة الأعداد الحقيقية ، والتى تمثل كميات صحيحة ( مثل 1.0 ) فى الصورة الصحيحة ( أى 1 ) . ويمكن أن تسبق أى عدد إشارة سالب أو إشارة موجب ، على ألا يكون هناك فراغ بين الإشارة والعدد .

ويجب معاملة البيانات الحرفية بطريقة مختلفة بعض الشئ ، حيث إن كل الرموز معنوية . ويجب ألا يوضع عنصر البيانات الحرفى بين علامتى تنصيص بصفة خاصة . كما أنه لايمكن فصل عنصر البيانات الحرفى عن عنصر البيانات الذى يسبقه باستخدام فراغ أو مؤشر نهاية السطر ، حيث إن هذه الفواصل تفسر ( بطريق خاطئ ) على أنها عناصر بيانات . ( فيفسر مؤشر نهاية السطر على أنه فراغ إذا ماحدد لمتغير حرفى ) .

### مـثال (٤ -٥)

؟ اعتبر مرة أخرى جزء برنامج البسكال الموجود فى المثال السابق ، أى :

```
VAR a,b : real;
    i,j : integer;
    p,q : char;
     .
     .
     .
read(a,b,i,j,p,q);
```

افرض أن القيم التالية سوف تحدد للمتغيرات

| Variable | Value |
|---|---|
| a | 12500 |
| b | −14.8 |
| i | 5 |
| j | −9 |
| p | X |
| q | Y |

يمكن إدخال عناصر البيانات فى ملف المدخلات على النحو التالى :

```
12500.0 -14.8 5 -9XY
```

أو يمكن إدخالها على النحو التالى :

```
12500 -14.8 +5 -9XY
```

أو على النحو التالى :

```
1.25e4 -1.48e1 5 -9XY
```

إذا ماظهرت عبارتان read وراء بعضهما ، فإن العبارة الثانية تبدأ من حيث انتهت العبارة الأولى . وبدقة أكثر... سوف تبدأ عبارة read الثانية بقراءة عنصر البيانات الذى يتبع آخر عنصر بيانات تم قراءته بعبارة read السابقة . وعلى هذا ... فإن عبارة read الجديدة لايلزم لها أن تبدأ بقراءة سطر جديد من أسطر البيانات .

### مــثال (٤ -٦)

دعنا نعتبر برنامج البسكال المذكور فى كل من المثالين السابقين . افرض أن عبارة read الموجودة فى هذا البرنامج استبدلت بالعبارتين التاليين :

```
read(a,b,i);
read(j,p,q);
```

لاتكون هناك حاجة إلى تغيير ملف المدخلات ، حيث إن أول عبارة read تقرأ أول ثلاث قيم موجودة فى السطر ، وثانى عبارة read تقرأ آخر ثلاث قيم موجودة فى السطر .

## ٣ – عبارة READLN : 3. THE READLN STATEMENT

تستخدم عبارة readln مثل عبارة read فى قراءة عناصر بيانات من ملف مدخلات . وتحدد هذه العناصر لمتغيرات صحيحة أو حقيقية أو حرفية . وتكتب هذه العبارة على النحو التالى :

readln(*input variables*)

والفرق بين العبارتين يقع فى أن عبارة readln تتسبب فى أن تبدأ read التالية ( وليست read الحالية ) فى بدء القراءة من سطر بيانات جديد ، بينما تسمح عبارة read لعبارة read أو readln التالية بأن تبدأ بالقراءة من نفس السطر .

### مــثال (٤ -٧)

فيما يلى جزء من برنامج بسكال

```
VAR p1,p2,p3,p4 : integer;
     .
     .
     .
read(p1,p2);
read(p3,p4);
```

افرض أن ملف المدخلات يحتوى على الأعداد الثمانية التالية ، مرتبة فى سطرين :

```
1 2 3 4
5 6 7 8
```

تتسبب عبارات read فى تحديد الأرقام 1 , 2 , 3 , 4 للمتغيرات p1 , p2 , p3 , p4 على التوالى .

أما إذا استبدلت عبارات read بعبارات readln ، أى يأخذ جزء البرنامج الشكل التالى :

```
VAR p1,p2,p3,p4 : integer;
      .
      .
      .
readln(p1,p2);
readln(p3,p4);
```

فسوف يتحدد 1 , 2 للمتغيرين p1 , p2 ، كما فى الحالة السابقة . أما p3 , p4 ، فسوف يتحدد لهما 5 , 6 على التوالى ( وذلك من بيانات مدخلات السطر الثانى ) .

لاحظ أن أول عبارة readln هى التى تسببت فى تحديد قيم p3 , p4 من السطر الثانى من أسطر المدخلات .

إذا ماسبقت عبارة readln عبارة read ، فسوف تبدأ عبارة readln عند النقطة التى تنتهى عندها عبارة read . وعلى هذا ... لايلزم لعبارة readln أن تبدأ بقراءة سطر بيانات جديد . ( لاحظ أن عبارة readln تتصرف فى هذا الموقف مثل عبارة read تماما ) .

## مثال (٤ -٨)

افرض أن عبارات read فى المثال رقم ٤ - ٧ كتبت على النحو التالى :

```
read(p1,p2);
readln(p3,p4);
```

فى هذه الحالة تتحدد القيم 1 , 2 , 3 , 4 للمتغيرات p1 , p2 , p3 , p4 على التوالى ، حيث إن عبارة readln تبدأ بقراءة القيمة الثالثة من أول سطر بيانات . وعلى أية حال ... فإن عبارة read أو عبارة readln التالية تبدأ بقراءة 5 من السطر الثانى من أسطر البيانات .

وعبارة readln مريحة فى قراءة البيانات سطرا بسطر ، كما فى حالة تشغيل الدفعات المعتاد .

## مثال (٤-٩)

يحتوى أحد برامج البسكال على عبارات readln التالية :

```
readln(a,b);
readln(c,d);
readln(e,f);
```

افرض أن جميع المتغيرات من النوع الصحيح ، وأن ملف المدخلات يحتوى على الأعداد التالية :

```
1  2  3
4  5  6
7  8  9
```

سوف تتحدد القيم 4 , 5 للمتغيرين c , d ، والقيم 7 , 8 للمتغيرين e , f .

وليس من الواضح أى قيم سوف تتحدد للمتغيرين b , a ، حيث إنه غير معروف أين تبدأ أول عبارة readln . فإذا مابدأت العبارة فى بداية السطر ، فسوف تتحدد القيم 1 , 2 للمتغيرين b , a على التوالى . ومن ناحية أخرى ... إذا ماسبق هذه العبارة عبارة read ، ولتكن

```
read(x);
```

فسوف يتحدد 1 للمتغير x ، ويتحدد 2 , 3 للمتغيرين b , a على التوالى .

## ٤ – دالّتا EOF , EOLN : 4. THE EOLN AND EOF FUNCTIONS

يحتوى البسكال على دالتين قياسيتين ، هما eof , eoln . وهما مفيدتان فى قراءة بيانات المدخلات . وأول هاتين الدالتين eoln تعود بقيمة بوليان true إذا ماانتهى السطر الذى تقرأ منه البيانات ، وإلا فإنها تعود بقيمة false .

وتوضع قيمة eoln مساوية false بصورة تلقائية عندما تظهر بداية سطر بيانات جديد ، بافتراض أن السطر التالى ليس خاليا ( أى أن السطر التالى يجب أن يكون محتويا على عنصر بيانات واحد على الأقل ) . وعلى هذا ... فتسمح هذه الدالة بقراءة عدد غير محدد من عناصر البيانات من أى سطر ، وذلك بقراءة البيانات من نفس السطر ، حتى تعود eoln بقيمة true . ويمكننا عند ذلك أن نعيد هذا الاجراء بالنسبة للسطر الثانى من أسطر البيانات ، وهكذا . وسوف توضح الطريقة الدقيقة التى يحدث بها ذلك فيما بعد فى هذا الكتاب ( فى الفصل السابع ، والفصل التاسع ، والفصل العاشر ) .

وعادة ماتستخدم دالة eof لاكتشاف نهاية ملف المدخلات ( أى تحديد نهاية الملف ) . وتعود هذه الدالة بقيمة بوليان true عندما تكتشف نهاية الملف ، وإلا فإنها تعود بقيمة false . وسوف يوضح استخدام دالة eof فى الفصل الحادى عشر من الكتاب .

وعادة ماتستخدم دالة eof مع دالة eoln لقراءة عدد غير محدد من عناصر البيانات من ملف المدخلات . ويمكن اكتشاف انتهاء كل سطر بواسطة دالة eoln ، كما يمكن اكتشاف انتهاء الملف ( أى انتهاء آخر سطر ) باستخدام دالة eof . وسوف نرى كيف يمكن تحقيق ذلك فى الفصل الحادى عشر .

## ٥ – عبارة اكتب : 5. THE WRITE STATEMENT

تستخدم عبارة ( اكتب ) لكتابة عناصر بيانات فى ملف المخرجات ، وتكتب هذه العبارة على النحو التالى :

write(*output data items*)

ويمكن لعناصر بيانات المخرجات أن تكون سلاسل أو ثوابت عددية ، أو قيما لمتغيرات أو تعبيرات . ويمكن أن تكون من النوع الصحيح أو الحقيقى أو الحرفى أو البوليانى . ( ويجب أن توضع كل سلسلة بين علامتى تنصيص ، كما سبق ذكره فى الفصل الثانى من الكتاب ) . ويجب أن تفصل عناصر البيانات بواسطة فواصل ، إذا ماكان هناك أكثر من عنصر بيانات واحد .

### مـثال (٤–١٠)

فيما يلى مثالاً نمطيًا لعبارة اكتب

```
write('x=',x);
```

تتسبب هذه العبارة فى كتابة القيمة العددية للمتغير x فى ملف المخرجات ، وذلك مع الاسم المناظر لها . فإذا ماكانت x تمثل 123.456 ، فسوف يكتب مايلى فى ملف المخرجات .

```
x= 1.2345600E+02
```

وبالمثل فإن عبارة اكتب التالية :

```
write('sum=',a+b);
```

تتسبب فى كتابة القيمة العددية لناتج التعبير a+b فى ملف المخرجات مع الاسم المناظر لها . وعلى هذا ... فإذا كان b , a يمثلان 3 , 1- على التوالى ، فسوف ينتج عن عبارة اكتب المخرجات التالية :

```
sum=          2
```

ويمكن عرض الأعداد الحقيقية بعدة طرق مختلفة ، بالرغم من أن الطريقة النمطية هى التمثيل العلمى ، كما هو موضح فى المثال السابق . وتمثل عناصر بيانات بوليان بمعرفات قياسية truc ، أو false طبقا لقيمها .

وتستخدم معظم صيغ البسكال عرضا قياسيا للحقل field width ( أى عدد الخانات التى يشغلها الحقل ) لتمثيل عناصر البيانات الصحيحة والحقيقية والبوليان . وسوف يختلف عرض الحقل هذا لكل نوع من أنواع البيانات . والأكثر من هذا ... فإن عرض الحقل المستخدم لنوع بيانات معين يتغير من صيغة لصيغة أخرى من صيغ البسكال . وبصورة تقليدية يمكن أن يكون للبيانات الصحيحة عرض قياسى للحقل ، مكون من 8 خانات . ويمكن أن يكون للبيانات الحقيقية عرض قياسى للحقل ، مكون من 14 خانة ، منها 7 خانات على يمين العلامة العشرية . ويمكن أن يكون لبيانات بوليان عرض قياسى مكون من 6 خانات ، فإذا ماكان الحقل أعرض من اللازم ( كما هو الحال فى العادة ) ، يوضع عنصر البيانات فى الجزء الأيمن من الحقل . وينتج عن هذا وجود فراغ واحد أو أكثر يسبق البيانات كما هو موضح في المثال السابق .

ويمكن أن يتغير العرض القياسى للحقل بسهولة ليسمح للمبرمج بتحكم أكبر فى شكل عناصر المخرجات . وطريقة عمل ذلك موضحة فيما بعد فى هذا الفصل .

وعند كتابة سلسلة أو بيانات من النوع الحرفى كمخرجات ، يكون عرض الحقل مساويا بالضبط لعدد خانات عنصر البيانات . وعلى هذا ... فلن تكون هناك أى فراغات تسبق أو تتبع عنصر البيانات ، إلا إذا كانت هذه الفراغات جزءا من البيانات الفعلية .

### مـثال (٤-١١)

يحتوى أحد برامج البسكال على عبارة اكتب التالية :

```
write('RED',' WHITE',' BLUE');
```

سوف ينتج مايلى عن هذه العبارة

```
RED WHITE BLUE
```

لاحظ أن الفراغ الذى يفصل الكلمات الفردية موجود فى آخر سلسلتين . أما إذا كانت عبارة اكتب على النحو التالى :

```
write('RED','WHITE','BLUE');
```

فسوف ينتج عنها مايلى :

```
REDWHITEBLUE
```

إذا ماتبع عبارة اكتب عبارة اكتب أخرى ، فتبدأ عبارة اكتب الثانية من عند انتهاء العبارة الأولى . وفى كلمات أخرى ... فإن أول عنصر بيانات يكتب بواسطة عبارة write الثانية يبدأ فى السطر الحالى بعد آخر عنصر بيانات كتب بواسطة عبارة write الأولى مباشرة . وعلى هذا ... فلايلزم لعبارة write جديدة أن ينتج عنها سطر بيانات مخرجات جديد .

مـثال (٤-١٢)

فيما يلى جزءًا من أجزاء برنامج بسكال

```
VAR a,b : real;
    i,j : integer;
    p,q : char;
     .
     .
     .
write(' a=',a,' b=',b,' i=',i);
write(' j=',j,' p=',p,' q=',q);
```

افرض أن القيم التالية حددت للمتغيرات :

| Variable | Value |
|---|---|
| a | 12500.0 |
| b | −14.8 |
| i | 5 |
| j | −9 |
| p | X |
| q | Y |

وعلى هذا تظهر بيانات المخرجات فى سطر واحد على النحو التالى :

```
a= 1.2500000E+04 b=-1.4800000E+01 i=         5 j=        -9 p=X q=Y
```

## ٦ – عبارة WRITELN : 6. THE WRITELN STATEMENT

عبارة writeln متطابقة مع عبارة write ، فيما عدا أنها ينتج عنها تحديد نهاية للسطر بعد كتابة آخر عنصر بيانات . وعلى هذا ... فإن عبارة write ، أو عبارة writeln تليها تبدأ فى سطر جديد .

وتكتب العبارة على النحو التالى :

```
writeln(output data items)
```

حيث يمكن أن تكون عناصر بيانات المخرجات سلاسل أو ثوابت عددية ، أو قيم لمتغيرات أو تعبيرات من النوع الصحيح أو الحقيقى أو البوليان .

## مـثال (٤-١٣)

افرض أن عبارات write الموجودة فى المثال السابق استبدلت بعبارات writeln ، أى

```
VAR a,b  : real;
    i,j  : integer;
    p,q  : char;
      .
      .
      .
writeln(' a=',a,' b=',b,' i=',i);
writeln(' j=',j,' p=',p,' q=',q);
```

وعلى هذا ينتج سطرا المخرجات التاليان :

```
a= 1.2500000E+04 b=-1.4800000E+01 i=        5

j=      -9 p=X q=Y
```

لاحظ أن أول عبارة writeln هى التى تسببت فى تجزئة السطر الفردى الأصلى للمخرجات إلى سطرين . أما ثانى عبارة write ، فليس لها أى تأثير على المخرجات ، بالرغم من أن أى مخرجات تالية subsequent تبدأ فى سطر جديد . وعلى هذا ... فإن العباراتين التاليتين :

```
writeln(' a=',a,' b=',b,' i=',i);
write(' j=',j,' p=',p,' q=',q);
```

ينتج عنهما سطرا المخرجات الذى سبق ذكرهما ، أما العبارتان التاليتان :

```
write(' a=',a,' b=',b,' i=',i);
writeln(' j=',j,' p=',p,' q=',q);
```

فينتج عنهما سطر المخرجات الذى سبق ذكره فى المثال رقم ١٢ - ٤ .

ويمكن استخدام عبارة writeln فارغة empty ، وذلك لإنتاج سطر فارغ ، كما هو موضح فى المثال التالى .

## مـثال (٤-١٤)

اعتبر الثلاث عبارات writeln التالية :

```
writeln('line one');
writeln;
writeln('line two');
```

وينتج عن هذه العبارات ثلاثة أسطر مخرجات ( تشمل سطر فارغا ) ، كما هو موضح أدناه :

```
line one

line two
```

فإذا لم توجد عبارة writeln الفارغة ؛ فلن يظهر السطر الفارغ ، ويظهر السطران المطبوعان وراء بعضهما على النحو التالى :

```
line one
line two
```

## ٧ – المخرجات المشكلة : 7. FORMATTED OUTPUT

عادة مايمكن جعل بيانات المخرجات مضبوطة بتغيير عرض الحقول المصاحبة للبيانات العددية ، ولبيانات بوليان . ويمكن أن يتحقق ذلك بسهولة بإضافة معالم تشكيل format معينة داخل عبارات write ، وعبارات writeln ، وبصفة خاصة فإن كل عنصر حقيقى أو بوليان فى عبارة write أو عبارة writeln يمكن أن تتبعه فاصلة ، أو كمية صحيحة موجبة تحدد عرض الحقل . ويمكن التعبير عن هذه الكمية كثابت أو متغير أو تعبير .

### مـثال (٤-١٥)

يحتوى أحد برامج البسكال على العبارة التالية :

```
write('The answer is',sum : 4);
```

افرض أن sum متغير صحيح ، محدد له القيمة 16- وعلى ذلك تكتب القيمة باستخدام حقل عرضه 4 خانات ، يعطى المخرجات التالية :

```
The answer is -16
```

( لاحظ أن الفراغ الذى يسبق العدد ، هو جزء من عرض الحقل المكون من ٤ خانات ) .

### مـثال (٤-١٦)

افرض الآن أن عبارة write فى المثال السابق أخذت الشكل التالى :

```
write('The answer is',sum : j+2);
```

فإذا كان j متغيراً صحيحاً ، محدد له القيمة 3 ، فإن عرض حقل النتيجة المصاحبة للمتغير sum يكـون 5 خـانات . وتظهر المخرجات على النحو التالى :

```
The answer is  -16
```

( لاحظ أن هناك فراغين يسبقن العدد ) .

ويفسر الغرض المحدد للحقل بأنه أدنى minimum عرض حقل يصاحب عنصر البيانات . أما إذا كان الحقل كبيرا جدا ، فيرحل عنصر البيانات إلى يمين الحقل ، تاركا فراغا واحدا أو أكثر قبل العنصر . ومن ناحية أخرى يضاف فراغ إضافى بصورة تلقائية إذا ما كان الحقل صغيرا جدا . وهذا يمنع تبديل عناصر البيانات .

## مـثال (٤-١٧)

بالعودة إلى المثال رقم ٤ -١٥، افرض أنه تحدد قيمة 3 للمتغير sum . وعلى هذا .. تظهر المخرجات كمايلى

```
The answer is    3
```

( لاحظ أن العدد محصور فى الجزء الأيمن للحقل ، وينتج عن ذلك وجود ثلاثة فراغات بين is ، 3 ) .

ومن ناحية أخرى .. إذا ماتحدد قيمة 10000 للمتغير sum ، فإن المخرجات تظهر على النحو التالى :

```
The answer is 10000
```

وقد أضيف فراغان إضافيان لحقل المخرجات ، ليلائم عدداً من 5 خانات ، والإشارة التى تسبقه .

اعتبر الآن أنه تحدد قيمة 30000- للمتغير sum . بهذا تصبح المخرجات على النحو التالى :

```
The answer is-30000
```

ويمكن كتابة عدد من النوع الحقيقى بطريقة مريحة أكثر ( بدون الأس ) إذا ماكان هناك عنصر مشكل ثان . وعلى هذا ... فإن تحديد التشكيل الكامل لعدد حقيقى ، يكون عبارة عن فاصلة يتبعها رقم صحيح يحدد إجمالى عرض الحقل ، يتبعه فاصلة أخرى ، ويتبع هذه الفاصلة رقم صحيح آخر ، يحدد عدد المواقع الموجودة على يمين العلامة العشرية . وسوف يقرب الكسر العشرى للعدد ذاتيا عندما تكون هناك حاجة لذلك .

## مـثال (٤-١٨)

افرض أن root متغير حقيقى ، محدد له القيمة 17487.266 ، يمكن كتابة هذا العدد بعدة طرق مختلفة كما هو موضح أدناه :

| Writeln Statement | Resulting Output |
|---|---|
| `writeln('root=',root);` | `root= 1.7487266E+04` |
| `writeln('root=',root : 12);` | `root= 1.74873E+04` |
| `writeln('root=',root : 18);` | `root= 1.74872660000E+04` |
| `writeln('root=',root : 10 : 3);` | `root= 17487.266` |
| `writeln('root=',root : 8 : 1);` | `root= 17487.3` |

تحتوى لغة البسكال على عبارة صفحة page . وهى عملية قياسية تتسبب فى ظهور عنصر البيانات المطبوع التالى فى قمة صفحة جديدة .

## مـثال (٤-١٩)

افرض أن أحد برامج البسكال يحتوى على الثلاث عبارات التالية :

```
writeln(a,b,c);
page;
writeln(x,y,z);
```

تتسبب هذه العبارات فى طباعة قيم المتغيرات c , b , a فى سطر واحد ، وتتبعها قيم المتغيرات z , y , x فى سطر آخر . وتتسبب عبارة page فى طباعة سطر المخرجات الثانى فى بداية صفحة جديدة .

ولاتميز العديد من النظم المتداخلة عبارة page ( فهى تهمل ببساطة ) ، إلا أن عبارة page تتسبب – على أية حال – مع بعض النظم المتداخلة فى مسح محتويات الشاشة .

وتفسر بعض نظم الكمبيوتر أو ل خانة من كل سطر من أسطر المخرجات بأنها خانة التحكم فى الطابع printer control ، ولاتطبع محتويات هذه الخانة ، لكنها تستخدم للتحكم فى المسافات الموجودة بين أسطر الطباعة .

وفيما يلى رموز تحكم شائعة الاستخدام فى التحكم فى الطابع

| الرمز | تفسيره |
|---|---|
| فراغ | الطباعة تحدث بعد السطر السابق مباشرة ( مخرجات فردية المسافة بين الأسطر ) |
| 0 | يترك سطر فارغ ، ويكتب فى السطر التالى له ( مخرجات زوجية المسافة بين الأسطر ) |
| 1 | يترك صفحة فارغة قبل البدء فى الطباعة . |
| + | تتم الطباعة على نفس السطر السابق . |

## مثال (٤-٢٠)

يشمل أحد برامج البسكال عبارات writeln التالية :

```
writeln(a,b,c);
writeln('1',x,y,z);
```

تتسبب هذه العبارات فى طباعة قيم المتغيرات c , b , a على سطر واحد ، تتبعه قيم المتغيرات z , y , x على سطر آخر . ويظهر السطر الثانى فى بداية صفحة جديدة – على أية حال – وذلك بسبب وجود رمز التحكم فى الطابع فى عبارة writeln الثانية . وعلى هذا ... فسوف تظهر المخرجات بنفس الطريقة التى ظهرت بها فى المثال السابق .

أما إذا ما تغيرت عبارة writeln إلى الشكل التالى :

```
writeln('0',x,y,z);
```

فيظهر سطرا المخرجات فى نفس الصفحة ، مع وجود سطر فارغ بينهما ( تكون المخرجات مزدوجة المسافة ) .

ويجب أن يفهم أن هذه الرموز المستخدمة فى التحكم فى الطابع لاتميزها كل صيغ البسكال . ويجب أن يحدد القارئ ماإذا كانت هذه السمة ميسرة له فى الجهاز المتاح له أم لا .

## ٨ – ملاحظات أخيرة : 8. CLOSING REMARKS

قبل ترك هذا الفصل ، توجد هناك ملاحظات عامة يمكن ذكرها عن عبارات page , writeln , write , readln , read .

أولا : هذه هى فى الواقع إجراءات قياسية standard procedures موجودة كجزء من لغة البسكال . وقد ناقشنا بالفعل استخدام الإجراءات القياسية المختلفة ، وطريقة الاتصال بها فى الفصل الثانى من الكتاب . وعلى القارئ أن يتذكر مرة أخرى أن الاتصال بإجراء قياسى يعتبر نوعًا من أنواع العبارات البسيطة . وفى هذا المفهوم تشير إلى استخدام هذه الإجراءات كعبارات .

ثانيا : يجب أن يكون مفهوما أن هذه العبارات تمثل فى الواقع فئة جزئية بسيطة من إمكانات المدخلات والمخرجات فى البسكال . وسوف نعتبر موضوع عمليات المدخلات والمخرجات بتفصيل أكبر عند دراسة الملفات فى الفصل الحادى عشر من الكتاب .

وأخيرا يجب أن يقدر القارئ أن هذه العمليات للمدخلات والمخرجات ، بالإضافة إلى المادة التى سبق ذكرها فى الفصول السابقة تقدم لنا إمكانية كتابة برامج بسكال كاملة ، لكنها بسيطة . وسوف يحتوى الفصل القادم على تعليمات لتنظيم وكتابة وتشغيل مثل هذه البرامج .

## أسئلة للمراجعة : Review Questions

(١) ماهو ملف المدخلات ؟ كيف يتم إدخال عناصر البيانات فى ملف المدخلات ؟

(٢) ماهو ملف المخرجات ؟ ماذا يحدث لعناصر البيانات التى يتم إدخالها فى ملف المخرجات ؟

(٣) كيف ترتب عناصر البيانات داخل ملف المدخلات أو ملف المخرجات ؟

(٤) كيف يشار إلى ملف المدخلات وملف المخرجات فى برنامج البسكال ؟

(٥) ماهو الغرض من عبارة اقرأ read ؟

(٦) فى أى معنى يجب أن تناظر عناصر البيانات فى ملف المدخلات متغيرات فى عبارة read ؟

(٧) لخص قواعد وضع عناصر البيانات العددية والحرفية فى ملف المدخلات .

(٨) هل يمكن تشغيل بيانات بوليان بعبارة read ؟

(٩) كيف تقرأ عناصر البيانات من ملف مدخلات عندما تتبع عبارة read عبارة read أخرى ؟

(١٠) ماهو الغرض من عبارة readln ؟ وماهو الفرق بينها وبين عبارة read ؟

(١١) ماذا يحدث عندما تسبق عبارة readln عبارة read ؟ وماذا يحدث إذا ماسبقت عبارة readln عبارة readln أخرى ؟

(١٢) ماهو الغرض من دالة eoln ؟ وماهو نوع المعلومات الذى تقدمه هذه الدالة ؟

(١٣) ماهو الغرض من دالة eof ؟ وماهو نوع المعلومات الذى تقدمه هذه الدالة ؟ وماهو الفرق بين هذه الدالة ودالة eoln ؟

(١٤) فى كلمات بسيطة ... كيف يمكن استخدام دالتى eof , eoln لقراءة عدد غير محدد من عناصر البيانات من ملفات مدخلات .

(١٥) ماهو الغرض من عبارة write ؟ وماهو نوع عناصر المخرجات التى يمكن أن توجد فى هذه العبارة ؟

(١٦) هل يمكن أن توجد سلاسل داخل عبارة write ؟ وهل يمكن أن توجد بها ثوابت عددية ؟ وهل يمكن أن توجد بها تعبيرات ؟ قارن ذلك بعبارة read .

(١٧) كيف تظهر الأعداد الحقيقية فى العادة عندما تكتب بواسطة عبارة write ؟

(١٨) هل يمكن تشغيل بيانات من نوع بوليان بعبارة write ؟ قارن ذلك بعبارة read .

(١٩) ماذا يعنى عرض الحقل ؟ وماهو عرض الحقل النمطى المستخدم فى الكمبيوتر المتاح لك ؟

(٢٠) لخص القواعد المستخدمة فى تحديد أماكن عناصر بيانات المخرجات .

(٢١) كيف تكتب عناصر البيانات فى ملف المخرجات عندما تتبع عبارة write عبارة write أخرى ؟

(٢٢) ماهو الغرض من عبارة writeln ؟ وماهو الفرق بينها وبين عبارة write ؟

(٢٣) ماذا يحدث عندما تسبق عبارة writeln عبارة write ؟ وماذا يحدث عندما تسبق عبارة writeln عبارة writeln أخرى ؟

(٢٤) لماذا يمكن استخدام عبارة writeln فارغة فى برنامج البسكال ؟ أى نوع من أنواع العبارات التى يتبعها عبارة writeln الفارغة فى معظم الأحوال ؟

(٢٥) كيف يمكن تغيير عرض الحقل فى ملف المخرجات ؟ وعلى أى نوع من أنواع البيانات يحدث ذلك ؟

(٢٦) ماذا يحدث إذا كان أحد حقول المخرجات أعرض من اللازم بالنسبة لعنصر بيانات معين ؟ وماذا يحدث اذا ماكان الحقل أقل من اللازم ؟

(٢٧) كيف يظهر عدد حقيقى فى صورة عشرية قياسية ( أى بدون أس ) ؟

(٢٨) افرض أن أحد الأعداد الحقيقية يظهر فى غير الصورة العلمية ، ولم يكن الحقل عريضا بدرجة تكفى الكسر العشرى للعدد . هل سيحدث تقريب للكسر ، أم سيحدث حذف دون تقريب ؟

(٢٩) ماهو الغرض من عبارة page ؟ كيف تفسر بعض نظم الكمبيوتر المتداخلة هذه العبارة ؟

(٣٠) هل يمكن استخدام رموز التحكم فى الطابع على جهاز الكمبيوتر المتاح لك ؟ إذا كانت الإجابة بنعم ؛ فما هى الرموز المتاحة ؟ وماهو الغرض من كل رمز من هذه الرموز ؟

(٣١) هل وصف هذا الفصل كل إجراءات المدخلات والمخرجات الموجودة فى البسكال ؟ وضح ذلك .

## مسائل محلولة : Solved Problems

(٣٢) فيما يلى هيكلاً تخطيطياً لبرنامج بسكال ، مع التركيز على معالم المدخلات والمخرجات

```
PROGRAM sample(input,output);
CONST factor = 12345;
      flag = 'entry point';
VAR i1,i2 : integer;
    r1,r2,r3 : real;
    c1,c2 : char;
    b1 : boolean;
```

```
BEGIN
   .
   .
   .
   readln(i1,i2,r1,r2,c1);
   .
   .
   .
   writeln(' I1=',i1:4, ' I2=',i2:4);
   writeln;
   writeln(' R1=',r1:10:2, ' R2=',r2:10:2, ' R3=',r3:12:4);
   writeln;
   write(' FIRST CHAR IS ',c1,' SECOND CHAR IS ',c2);
   write(' TEST STATUS: ',b1:5);
   .
   .
   .
END.
```

سوف يقرأ هذا البرنامج سطرا واحدا من البيانات ( فيه خمسة عناصر بيانات ) من ملف مدخلات ، وسوف يكتب البرنامج ثلاثة أسطر بيانات فى ملفات المخرجات . وسوف يتم تسمية وتشكيل عناصر بيانات المخرجات ، كما أن المسافات بين أسطر المخرجات تكون مزدوجة .

(٣٣) تشير مواقف المشاكل التالية الى ثوابت ومتغيرات معرفة فى المسألة السابقة . اكتب عبارات مناسبة ، أو مجموعة عبارات لكل موقف من هذه المواقف .

(أ) اقرأ قيم المتغيرات i1 , r3 , c2 من سطر بيانات واحد

```
read(i1,r3,c2);
```

or

```
readln(i1,r3,c2);
```

or

```
read(i1);
read(r3);
read(c2);
```

(ب) اقرأ قيم المتغيرات c1,r2,r1,i1 من سطر واحد ، تتبعه قيم المتغيرات c2,r3,i2 من سطر آخر ،

```
readln(i1,r1,r2,c1);
readln(i2,r3,c2);
```

يمكن استبدال ثانى عبارة readln بعبارة read على النحو التالى :

```
read(i2,r3,c2);
```

(ج) اكتب قيم الثابتين على سطر واحد ، تليه قيم كل المتغيرات على سطر آخر . لاتضع أسماء للمخرجات ، واترك فراغا بين الثابتين . أترك سطرين فارغين بين سطرى المخرجات ( أى مسافة ثلاثية ) .

```
writeln(factor,' ',flag);
writeln;
writeln;
writeln(i1,i2,r1,r2,r3,c1,c2,b1);
```

يمكن استبدال آخر عبارة writeln بعبارة write التالية :

```
write(i1,i2,r1,r2,r3,c1,c2,b1);
```

(د) اكتب قيم الثابتين على سطر واحد ، تليه قيم c1,r2,r1,i1 على سطر آخر ، يليه قيم c2,b1,r3,i2 على سطر آخر . ابدأ فى بداية صفحة جديدة ، مع ترك مسافة مزدوجة بين أسطر المخرجات . ضع أسماء لكل عناصر المخرجات .

```
page;
writeln('Factor=',factor,' Flag=',flag);
writeln;
writeln('I1=',i1,' R1=',r1,' R2=',r2,' C1=',c1);
writeln;
writeln('I2=',i2,' R3=',r3,' B1=',b1,' C2=',c2);
```

إذا مااستخدمت إحدى صيغ البسكال الموجود بها رموز التحكم فى الطابع ، فيمكن كتابة العبارات السابقة على النحو التالى أيضا :

```
writeln('1Factor=',factor,' Flag=',flag);
writeln('0I1=',i1,' R1=',r1,' R2=',r2,' C1=',c1);
writeln('0I2=',i2,' R3=',r3,' B1=',b1,' C2=',c2);
```

(هـ) أعد المسألة السابقة ، ولاتضع فى هذه الحالة أسماء لبيانات المخرجات ، ولكن شكل بيانات المخرجات على النحو التالى :

integer: 5 characters
real: 10 characters, with 3 digits to the right of the decimal point (no exponent)
boolean: 7 characters

كما هو مطلوب أن يسبق كل عنصر بيانات حرفى فراغ واحد

```
page;
writeln(factor:5,' ',flag);
writeln;
writeln(i1:5, r1:10:3, r2:10:3,' ',c1);
writeln;
writeln(i2:5, r3:10:3, b1:7,' ',c2);
```

يمكن اعادة كتابة هذه العبارات باستخدام خانات تحكم الطابعة كما يلى

```
writeln('1',factor:5,' ',flag);
writeln('0',i1:5, r1:10:3, r2:10:3,'  ,c1);
writeln('0',i2:5, r3:10:3, b1:7,' ',c2);
```

(٣٤) افرض أن ملف المدخلات يحتوى على سطرى البيانات التاليين :

```
10  0.005  4.66E12ABC
-817  2.7E-3  XYZ
```

ماهى القيم التى تحدد للمتغيرات الموجودة فى السؤال السابق (ب) .

```
i1=10   r1=0.005   r2=4.66E+12   c1=A
i2=-817   r3=2.7E-03   c2= (blank space)
```

(٣٥) افرض أن أحد ملفات المدخلات يحتوى على الأربعة أسطر بيانات التالية :

```
10  0.005
4.66E12  ABC  -63  17.7
-75  33.9E  B
C
```

ماهى القيم التى تحدد للمتغيرات الموجودة فى السؤال الثانى (ب) ؟

```
i1=10   r1=0.005   r2=4.66E+12   c1= (blank space)
i2=-75   r3=33.9   c2=E
```

(٣٦) افرض أن المتغيرات الموجودة فى السؤال (٣٤) محدد لها القيم التالية :

| Variable | Value |
|---|---|
| i1 | −630 |
| i2 | 375 |
| r1 | 20.8 |
| r2 | −477300.0 |
| r3 | 0.000185 |
| c1 | $ |
| c2 | 5 |
| b1 | خطأ |

بين كيف تظهر البيانات فى ملف المخرجات إذا ماكتبت طبقا لعبارات writeln المعطاه فى السؤال الثانى (ج) .

```
12345 entry point
(blank line)
(blank line)
 -630      375 2.0800000E+01-4.7730000E+05 1.8500000E-04$5 false
```

(٣٧) أعد المسألة السابقة ، مفترضا أن بيانات المخرجات مكتوبة طبقا لعبارات writeln المعطاه فى السؤال الثانى (د) .

```
Factor=12345 Flag=entry point
(blank line)
I1=     -630 R1= 2.0800000E+01 R2=-4.7730000E+05 C1=$
(blank line)
I2=      375 R3= 1.8500000E-04 B1= false C2=5
```

(٣٨) أعد المسألة رقم ٣٦ ، مفترضا أن بيانات المخرجات مكتوبة طبقا لعبارات writeln المعطاه فى السؤال ٣٤ (هـ) .

```
12345 entry point
(blank line)
 -630     20.800-477300.000 $
(blank line)
  375      0.000   false 5
```

(٣٩) فيما يلي جزءًا من برنامج بسكال

```
PROGRAM sample2(input,output);
VAR a,b,c,d : integer;
BEGIN
    .
    .
    .
   readln(a,b,c,d);
    .
    .
    .
   writeln(5*(a+b)/2 : c+2, 10*(a-b)/2 : c+d-1);
    .
    .
    .
END.
```

افرض أن ملف المدخلات يحتوى على البيانات التالية :

3 5 4 1

ماهى المخرجات التى تنتج عن هذا البرنامج ؟

القيمتان العدديتان هما 20 , 10- وعرض الحقل المناظر لكل منهما هو 6 , 4 على التوالى . وعلى هذا تظهر المخرجات كمايلى :

```
    20 -10
```

مع وجود 4 فراغات تسبق أول عدد

## Supplementary Problems — مشاكل متكاملة :

(٤٠) فيما يلى هكيلاً تخطيطيا لبرنامج بسكال كامل :

```
PROGRAM example(input,output);
CONST flag = 'red';
      factor = 0.005;
VAR i1,i2,i3 : integer;
    r1,r2 : real;
    c1,c2,c3,c4 : char;
    b1,b2 : boolean;
BEGIN
    .
    .
    .
   read(i1,i2);
   readln(i3,r1,r2);
   read(c1,c2,c3,c4);
    .
    .
    .
   page;
   writeln(flag,factor);
   writeln;
   write(i1,i2);
   writeln(i3,r1,r2);
   writeln;
   write(c1,c2,c3,c4);
   writeln(b1,b2);
    .
    .
    .
END.
```

فى كلمات بسيطة ، كيف يجب إدخال عناصر بيانات المدخلات ؟ وكيف تبدو عناصر بيانات المخرجات ؟ هل تفصل عناصر المخرجات وتميز من بعضها ؟

(٤١) كل من المشاكل التالية يبين فئة من البيانات داخل ملف مدخلات . حدد فى كل حالة القيم التى تحدد المتغيرات الموجودة فى السؤال السابق .

(*a*)
```
1   2   3   4.0   5.0
blue   green
```

(*b*)
```
1
2
3
4
5
b
l
u
e
```

(*c*)
```
1   2   3   4.0   5.0   blue
```

(*d*)
```
1   2
3   4
5   blue
green
```

(٤٢) كيف تظهر المخرجات الناتجة من السؤال الأول ، مفترضا أن القيم التالية محددة للمتغيرات ؟

| Variable | Value |
|---|---|
| i1 | 100 |
| i2 | -200 |
| i3 | -300 |
| r1 | 400.444 |
| r2 | -500.555 |
| c1 | P |
| c2 | I |
| c3 | N |
| c4 | K |
| b1 | true |
| b2 | false |

(٤٣) افرض أن عبارات المخرجات في السؤال الأول استبدلت بالعبارات التالية :

```
page;
writeln('flag=',flag,' factor=',factor:6:3);
writeln;
write('i1=',i1:4,' i2=',i2:4);
writeln('i3=',i3:4,' r1=',r1:6:1,' r2=',r2:6:1);
writeln;
write('color=',c1,c2,c3,c4);
writeln('b1=',b1:5,' b2=',b2:5);
```

كيف تبدو المخرجات ، مفترضا أن القيم الموجودة في السؤال السابق حددت للمتغيرات ؟

(٤٤) افرض الآن أن صيغة البسكال المستخدمة تستخدم رموز تحكم في الطابع ، وأن مخرجات السؤال الأول استبدلت بما يلي :

```
writeln('1flag=',flag,' factor=',factor:6:3);
writeln('0i1=',i1:5,' i2=',i2:5,' i3=',i3:5);
writeln(' r1=',r1:8:2,' r2=',r2:8:2);
writeln('1color=',c1,c2,c3,c4);
writeln('0b1=',b1:6,' b2=',b2:6);
```

كيف تبدو المخرجات ، مفترضا أن القيم الموجودة في السؤال الثالث حددت للمتغيرات ؟

(٤٥) مواقف المشاكل التالية تشير إلى ثوابت ومتغيرات معرفة في السؤال الأول . اكتب العبارات المناسبة ، أو مجموعة العبارات لكل من هذه المواقف .

(أ) اقرأ كل عناصر بيانات المدخلات من سطر بيانات واحد .

(ب) اقرأ قيم i1 , r2 , r3 , c4 من سطر بيانات واحد .

(ج) اقرأ عناصر البيانات الصحيحة من سطر بيانات واحد ، وعناصر البيانات الحقيقية من سطر آخر ، وعناصر البيانات الحرفية من سطر آخر .

(د) اقرأ كل عنصر بيانات من سطر منفصل .

(هـ) أكتب قيم الثابتين وكل المتغيرات في سطر واحد . لاتضع أسماء للمخرجات ، ولكن اترك فراغا واحدا على الأقل بين كل عنصري بيانات .

(و) أعد المسألة السابقة ، مع تشكيل البيانات العددية وبيانات بوليان طبقا لمايلى :

الصحيح : 4 خانات .

الحقيقى : 8 خانات ، منها اثنان على يمين العلامة العشرية ( بدون استخدام أس ) .

البوليان : 6 خانات

(ز) أكتب قيمتى الثابتين على سطر واحد ، تتبعه قيم المتغيرات الحرفية ، ومتغيرات بوليان على سطر آخر ، تتبعه قيم المتغيرات العددية على سطر ثالث . لاتضع أسماء للمخرجات ، واترك فراغا واحدا على الأقل بين كل عنصرى بيانات . اترك سطرا فارغا أيضا بين كل سطر من أسطر الطباعة ( أى أن الطباعة تكون مزدوجة المسافة ) .

(ح) أعد المسألة السابقة ، مع تسمية كل ثابت ، وكل قيمة عددية ، وكل عنصر بيانات بوليان . اكتب البيانات الحرفية كرموز متتالية ، بدون ظهور أى فرغات مع ظهور اسم واحد يسبق هذه الرموز .

(ط) اكتب قيم b1 , c2 , c1 , r1 , i1 , flag على سطر واحد ، تتبعه قيم b2 , c4 , c3 , r2 , i3 , i2 , factor على سطر آخر . لاتضع أسماء للمخرجات ، وضع فراغ واحد على الأقل بين كل عنصرى بيانات . أبدأ أول سطر فى بداية صفحة جديدة ، واترك سطرين فارغين بين أسطر المخرجات المطبوعة .

(ى) أعد المسألة السابقة ، مع تشكيل البيانات العددية ، وبيانات بوليان على النحو التالى :

صحيح : 5 خانات

حقيقى : 7 خانات ، اثنان منها على يمين العلامة العشرية ( لايوجد أس )

بوليان : 7 خانات .

(ك) اكتب القيم لكل من الثابتين وكل المتغيرات . ابدأ فى بداية صفحة جديدة ، واكتب كل عنصر بيانات على سطر مستقل . ضع أسماء لكل عناصر البيانات . شكل البيانات على النحو التالى :

صحيح : 4 خانات

حقيقى : 12 خانة ، مستخدما أس

بوليان : 5 خانات

(٤٦) فيما يلى جزءا من برنامج بسكال :

```
PROGRAM example2(input,output);
VAR w,x,y,z : integer;
BEGIN
   .
   .
   .
   readln(w,x,y,z);
   .
   .
   .
   writeln('SUM=',x+y+z : w,' PRODUCT=',x*y*z : w+3);
   .
   .
   .
END.
```

افرض أن ملف المدخلات يحتوى على القيم التالية :

3 10 20 30

بين كيف تكون المخرجات .

# الفصل الخامس

# إعداد وتشغيل برنامج بسكال كامل

# Preparing and Running a Complete Pascal Program

حتى الآن تعلمنا مايكفى من لغة البسكال لكتابة برنامج كامل وبسيط . وعلى هذا ... فلنلق بعض الانتباه على تخطيط وكتابة وتنفيذ مثل هذه البرامج . كما نأخذ فى الاعتبار أيضا أوجه البرمجة الخاصة بالبسكال ، مثل التطوير المنطقى للبرنامج ، وأسلوب البرمجة الجيد ، وعملية تطوير برنامج كامل .

## ١ – تخطيط برنامج البسكال : 1. PLANNING A PASCAL PROGRAM

من الضرورى رسم سياسة عامة للبرنامج قبل أن تبدأ أى تفاصيل برمجة فعلية . وبهذه الطريقة يستطيع أن يركز المبرمج أساسا على المنطق العام للبرنامج ، دون أن يدخل فى التفاصيل التكوينية للتعليمات الفردية . وقد تعاد عملية التخطيط الشامل هذه عدة مرات ، مع إضافة تفاصيل برمجة فى كل مرة تحدث فيها الإعادة . ويستطيع المبرمج على هذا أن ينقل انتباهه تدريجيا من سياسة الحسابات الشاملة إلى تفاصيل التعليمات الفردية . ويشار إلى مثل هذا الأسلوب بالبرمجة من القمة إلى القاعدة .

وعادة مايتم تنظيم البرنامج من القمة إلى القاعدة عن طريق إعداد تخطيط غير رسمى ، يحتوى على عبارات ( أو جمل ) كجزء من اللغة الإنجليزية وجزء من البسكال . وفى المرحلة الأولية تكون كمية البسكال قليلة جدا ، ولاتظهر سوى كلمات أساسية متعددة تعرف مكونات البرنامج الأساسية فقط ، مثل ( END , BEGIN , PROGRAM ) . وبعد ذلك تدخل المادة الوصفية باللغة الإنجليزية بين هذه الكلمات الأساسية . ويكون ذلك عادة على هيئة تعليقات فى البرنامج . وعادة مايشار إلى التخطيط الناتج بأنه شفرة شبيهة pseudo code .

### مـثال (٥ –١)

بيع الفطائر Pizza : يبيع أحد محلات بيع الفطائر ثلاثة أحجام من الفطائر ، وهى الحجم الصغير ( قطره 10 بوصة ) ، والحجم المتوسط ( قطره 12 بوصة ) ، والحجم الكبير ( قطره 16 بوصة ) . ويمكن شراء الفطيرة بدون إضافات ، أى بالجبن والصلصة فقط ، أو بإضافات مثل البصل والجمبرى وغيرها .

ويريد صاحب المحل أن يعد برنامج كمبيوتر يحسب سعر بيع الفطيرة إذا ماأعطى له حجمها والمكونات الإضافية . سعر البيع يساوى مرة ونصف إجمالى التكلفة ، طبقا لحجم الفطيرة وعدد المكونات الإضافية . وسوف يشمل إجمالى التكلفة تكلفة أعداد ثابتة ، وتكلفة متغيرة تتناسب مع حجم الفطيرة ، وتكلفة متغيرة إضافية لكل مكون إضافى زيادة . وبغرض التبسيط ، يفترض أن تكلفة كل مكون إضافى لوحدة المساحة متساوية .

اعتبر الآن الفطيرة التى قطرها d وبها n مكون إضافى . يحسب سعر بيع هذه الفطيرة على النحو التالى :

$$price = 1.5 * cost$$

حيث

$$cost = fixedcost + (basecost * area) + (n * extracost * area)$$

و

$$area = \frac{\pi d^2}{4}$$

وعلى هذا يمكن تحديد سعر بيع الفطيرة ببساطة إذا كانت التكاليف المختلفة معروفة ، وكذلك يكون القطر وعدد المكونات الإضافية معروفين أيضا .

لاحظ أنه يمكن قراءة مكونات التكلفة مع بداية كل حالة ، وذلك مع القطر وعدد المكونات الإضافية . كما يمكن أيضا تعريف مكونات التكلفة كثوابت داخل البرنامج . وسوف نتبع الطريقة الأخيرة . وسبب ذلك هو الافتراض أن البرنامج سوف ينفذ العديد من المرات باستخدام قيم مختلفة للمدخلات بالنسبة للقطر وعدد المكونات الإضافية مع استخدام نفس التكاليف .

يمكننا الآن كتابة التخطيط العام التالى للبرنامج :

(١) تعريف أو تحديد عناصر التكلفة المختلفة .

(٢) قراءة القطر للفطيرة وعدد المكونات الإضافية .

(٣) حساب مساحة الفطيرة .

(٤) حساب التكلفة الكلية ، وسعر البيع .

(٥) كتابة الإجابة النهائية ( سعر البيع ) مع كتابة مدخلات كافية لتعريف المشكلة .

فإذا ماكتبنا هذا التخطيط بالشفرة الشبيهة ، فيمكننا الحصول على مايلى :

```
(* define the cost components as program constants *)

BEGIN

   (* read the diameter and the number of extra ingredients *)

   (* calculate the area of the pizza, the overall cost and the selling price *)

   (* write out the input data and the selling price *)

END.
```

وحيث إن هذه المشكلة بسيطة بصفة خاصة ، فلا تكون هناك حاجة إلى إدخال تعديلات إضافية على هذا التخطيط . أما إذا كانت المشكلة أكثر تعقيدا ، فيجب أن نكتب صيغا تفصيلية أكثر بالشفرة الشبيهة ، موضحين تفاصيل أدق للمنطق الشامل للبرنامج .

وهناك طريقة أخرى تستخدم فى بعض الأحيان فى تخطيط برنامج البسكال ، وهى أسلوب : من القاعدة إلى القمة . ويمكن أن تكون هذه الطريقة مفيدة فى حالة البرامج التى تستخدم برامج فرعية داخل البرنامج ( أى تستخدم دوال وإجراءات يعدها المستفيد ) والتى يمكن الاتصال بها من أجزاء أخرى من أجزاء البرنامج . وفى أسلوب من القاعدة إلى القمة يتم تطوير هذه البرامج الفرعية للبرنامج بالتفصيل أولا ، وذلك فى مرحلة مبكرة من عملية التخطيط الشامل . ويمكن بعد ذلك أن يعتمد أى تطوير للبرنامج على الخواص المعروفة لهذه الأجزاء من البرنامج ، أو هذه البرامج الفرعية . وعادة مايستخدم كلا من الأسلوبين عمليا : أسلوب من القاعدة إلى القمة لإعداد البرامج الفرعية قبل إعداد الجزء الرئيسى من البرنامج ، وأسلوب من القمة للقاعدة بالنسبة لإعداد كل برنامج فرعى . ويتم إعداد الجزء الرئيسى من البرنامج باستخدام أسلوب من القمة للقاعدة فى معظم الأحوال .

## ٢ - كتابة برنامج بسكال : 2. WRITING A PASCAL PROGRAM

بعد صياغة سياسة عامة للبرنامج ، وكتابة تخطيط البرنامج ، يمكن توجيه الاتجاه إلى تفاصيل عمل برنامج البسكال . وعند هذه النقطة يصبح التركيز على ترجمة كل خطوة من خطوات تخطيط البرنامج ( أو كل جزء من أجزاء الشفرة الشبيهة ) إلى إحدى تعليمات البسكال ، أو عدة تعليمات بسكال . ويجب أن يكون هذا النشاط نشاطا مباشرا ، مع افتراض أن سياسة البرنامج العامة تم إعدادها بدقة وبتفاصيل كافية .

وعند كتابة برنامج بسكال كامل يوجد - على أية حال - عدة نقاط يجب تذكرها دائما . أولا : يجب إدخال التوضيحات والتعريفات المختلفة بالترتيب المناسب . ثانيا : يجب إعطاء بعض الاهتمام بالسمات التى سوف تكون متاحة للمستفيد ( بالرغم من أن بعض هذه السمات يجب أن تؤخذ فى الاعتبار فى مرحلة تخطيط البرنامج ) . وأخيرا يجب الاهتمام بأوجه معينة لأسلوب البرمجة ، والتى تؤخذ فى الاعتبار عند تنظيم البرنامج ليكون واضحا ومعبرا . ولنأخذ كل من هذه النقاط بشئ من التفصيل .

سبق أن قدمنا فى الفصل الأول تخطيطا يوضح الهيكل الشامل لبرنامج البسكال . وفيما يلى اعادة لهذا التخطيط ، يليه بعض التعليقات التوضيحية :

1. Header
2. Block
   (*a*) Declarations
      - Labels
      - Constants
      - Type definitions
      - Variables
      - Procedures and functions
   (*b*) Statements

تذكر أن العنوان عبارة عن عنصر من سطر واحد ، يبدأ بكلمة PROGRAM ، ويتبعه اسم البرنامج وأدلة لأى ملفات مدخلات أو مخرجات . وتوضع أدلة الملفات بين قوسين ، وتنفصل عن بعضها بواسطة فواصل .

وتحتوى كتلة البرنامج ( المجموعة الرئيسية للبرنامج ) على جزء توضيحات ، وجزء لعبارات إجراءات . ويجب تقديم التوضيحات بالترتيب المطلوب ، وذلك بالرغم من أن أنواع التوضيحات المختلفة ليست فى حاجة إلى أن تظهر فى كل برنامج من برامج البسكال . وفى واقع الأمر ، فإن الثوابت والمتغيرات هى أنواع التوضيحات الوحيدة التى تؤخذ فى الاعتبار حتى الآن فى برنامج البسكال . أما النقطة الهامة الآن ، فهى أن نتذكر أن تعريفات الثوابت يجب أن تسبق توضيحات المتغيرات .

ومن الناحية التكوينية ، يسمح بعمق كبير فى كتابة عبارات الإجراءات . والمتطلب الثابت الوحيد هو أن توجد عبارات الإجراءات داخل عبارة شاملة مركبة ( أى بين END , BEGIN ) . وعلى هذا ... فيجب أن يحتوى كل برنامج على تسلسل BEGIN ، تليه بقية العبارات ، وينتهى بكلمة END . ويمكن أن توجد بقية العبارات المركبة داخل هذه العبارة المركبة الشاملة إذا كانت هناك رغبة فى ذلك . ويمكن أن تحتوى كل عبارة مركبة على عبارات إجراءات مختلفة ، أو على مجموعات عبارات ( وهناك المزيد عن ذلك فى الفصل القادم ) .

يجب إعطاء بعض الاهتمام بأمور التنقيط ، وبصفة خاصة باستخدام الفواصل المنقوطة . وتستخدم القواعد التالية فى تحقيق ذلك .

(١) تستخدم الفاصلة المنقوطة فى البسكال كفاصل Separator ( وليس كمحدد لنهاية terminator ) . وعلى هذا فإنها تستخدم بين عبارتين متتاليتين أو توضيحين متتالين ، بدلا من استخدامها فى نهاية كل عبارة أو كل توضيح .

(٢) تستخدم قواعد خاصة مع END , BEGIN . فهذان العنصران هما فى واقع الأمر قوسان brackets يحددان بداية ونهاية عبارة مركبة . وعلى هذا ... فلاتحتاج BEGIN لأن تليها فاصلة منقوطة ، كما لاتحتاج END لأن تسبقها فاصلة منقوطة .

(٣) سوف تفسر الفاصلة المنقوطة غير المطلوبة ( مثل التى تسبق END ) بأنها عبارة صفرية null statement ( أى عبارة خالية ) .

وعادة مالايكون لهذا تأثير ملحوظ على تنفيذ البرنامج ، بالرغم من أنه هناك مواقف معينة يمكن أن يتغير فيها منطق البرنامج ... دون أن يكون هذا مطلوبا . وعلى هذا ... فيجب تجنب ظهور الفواصل المنقوطة غير المطلوبة .

(٤) يجب ان ينتهى كل برنامج كامل بنقطة . وعلى هذا ... فان آخر END فى البرنامج يجب أن يتبعها نقطة .

ورسومات بسكال التكوينية الموجودة فى نهاية هذا الكتاب يمكن فحصها للحصول على معلومات دقيقة عن استخدام التنقيط مع عبارات محددة .

## مثال (٥ -٢)

**بيع الفطائر** : افرض أن مشكلة تسعير الفطائر المذكورة فى المثال السابق تستخدم بيانات التكلفة التالية :

تكلفة ثابتة = 0.75 $ لكل فطيرة

تكلفة أساسية = 0,01 $ لكل بوصة مربعة .

تكلفة كل عنصر إضافى = 0.0025 $ لكل بوصة مربعة .

وفيما يلى برنامج بسكال أولى يحتوى على عناصر التكلفة هذه :

```
PROGRAM pizzas(input,output);
CONST pi = 3.14159;
      fixedcost = 0.75;
      basecost = 0.01;
      extracost = 0.0025;
VAR n : integer;
    d,area,cost,price : real;
BEGIN
   readln(d,n);
   area := pi*sqr(d)/4;
   cost := fixedcost + basecost*area + n*extracost*area;
   price := 1.5*cost;
   writeln(d,n,price)
END.
```

لاحظ أن البرنامج يحتوى على تعريفات ثوابت وتوضيحات متغيرات ، وتسبق تعريفات الثوابت توضيحات المتغيرات . لاحظ أيضا أن التعريفات والتوضيحات والعبارات مفصولة بواسطة فواصل منقوطة ، مع ظهور الفواصل المنقوطة فى نهاية كل سطر عندما تكون هناك حاجة لذلك . ولاتتبع كلمة BEGIN فاصلة منقوطة ، كما لاتسبق كلمة END فاصلة منقوطة ، حيث إن هاتين الكلمتين الأساسيتين تخدمان كأقواس تعرف بداية ونهاية عبارة مركبة واحدة . لاحظ أخيرا النقطة الموجودة فى نهاية البرنامج كما هو مطلوب تماما .

والبرنامج السابق هو برنامج بسكال كامل ، إلا أنه ينقصه بعض السمات المرغوب فيها . وسوف تقدم صيغتان أكثر اكتمالا ( أكثر رغبة فى الحصول عليهما ) فى المثالين التاليين .

يجب أن يفهم القارئ أنه هناك المزيد لكتابة برنامج بسكال كامل عن ترتيب توضيحات وعبارات ببساطة فى ترتيبها الصحيح ، ووضع الفواصل والنقط بطريقة صحيحة . يجب توجيه الاهتمام أيضا إلى وجود سمات خاصة تحسن من قراءة البرنامج ، ومن نتائج مخرجاته . وتشمل هذه السمات التسلسل المنطقى للعبارات ، واستخدام الترحيل ، واستخدام التعليقات ، وإنتاج مخرجات بأسماء مناسبة .

ويتحدد تسلسل العبارات المنطقى داخل البرنامج إلى حد كبير ، وذلك بتحديد منطق البرنامج . وعادة مايكون هناك – على أية حال – طرق مختلفة عديدة لتسلسل عبارات معينة ، دون تغيير فى منطق البرنامج . وهذا صحيح بصفة خاصة بالنسبة للبرامج الأكثر تعقيدا ، والتى تحتوى على استخدام أجزاء برنامج شرطية أو متكررة . وفى مثل هذه الحالات يمكن أن يكون لتسلسل عبارات معينة أو مجموعة عبارات تأثير رئيسى على وضوح منطق البرنامج . ولذلك فمن المهم أن تكون العبارات متسلسلة فى أفضل طريقة فعالة . وسوف نذكر المزيد عن ذلك فى الفصل القادم عند مناقشة أنواع الشروط المختلفة ، ومعالم التكرار المتاحة فى البسكال .

ويرتبط استخدام الترحيل اتصالا وثيقا بتسلسل مجموعات العبارات داخل البرنامج . وبينما يؤثر التسلسل على ترتيب تنفيذ مجموعة العمليات ، فإن الترحيل يوضح طبيعة الأجزاء الخاصة بعبارات فردية داخل المجموعة . ومميزات الترحيل واضحة ، حتى بالنسبة للبرامج البسيطة التى سبق ذكرها فى هذا الكتاب . وسوف يصبح ذلك أكثر وضوحا فيما بعد ، كلما أخذنا فى الاعتبار برامج بسكال أكثر تعقيدا .

ويجب أن توجد تعليقات بصفة دائمة فى برنامج البسكال . وإذا ماكتبت مثل هذه التعليقات بطريقة مناسبة ، فيمكنها أن تقدم نظرة عامة مفيدة للمنطق العام لبرنامج . كما يمكنها أيضا أن تخطط الأجزاء الرئيسية للبرنامج ، وأن تعرف عناصر رئيسية معينة داخل البرنامج ، وأن تقدم معلومات مفيدة أخرى عن البرنامج . ( والتعليقات من هذا النوع يمكن أن تكون مفيدة جدا للمبرمج ، ولأى شخص آخر يحاول أن يقرأ البرنامج ويفهمه ، حيث ينسى المبرمجون فى بعض الأحيان تفاصيل برامجهم مع مرور الزمن . وهذا صحيح بصفة خاصة بالنسبة للبرامج الطويلة المعقدة ) . وعادة مالاتحتاج التعليقات التى توضع داخل البرنامج أن تكون طويلة وكثيرة ، فقلة من التعليقات التى توضع فى أماكن مناسبة يمكنها أن تلقى ضوءا كافيا على البرنامج .

وهناك صفة هامة أخرى للبرنامج الجيد ، وهى مقدرته على إنتاج مخرجات واضحة ومقروءة . ويسهم عاملان فى هذا الوضوح ، أولهما : هو تسمية بيانات المخرجات كما سبق ذكره فى الفصل الرابع من الكتاب . وثانيهما : هو ظهور بعض بيانات المدخلات مع المخرجات ، بحيث إن كل فئة من فئات بيانات المدخلات ( إذا ماكان هناك أكثر من فئة واحـدة ) يمكن تعريفها بدقة . وطريقة تحقيق ذلك تعتمد على البيئة التى ينفذ فيها برنامج البسكال .

ففى البيئة غير المتداخلة يجب أن تطبع كمية معينة من بيانات المدخلات ، وذلك بالإضافة إلى المخرجات المطلوبة . إذا كانت المدخلات كبيرة ، فيجب عرض عدد بسيط من العناصر الرئيسية لبيانات المدخلات تكفى لتعريف كل مشكلة خاصة ( أى لتعريف كل فئة من فئات بيانات المخرجات ) .

### مـثال (٥ –٣)

نقدم الآن صيغة مطورة من برنامج تسعير الفطائر الذى سبق ذكره فى المثال السابق . وتشمل الصيغة الحالية تعريف تعليقات وإنتاج بيانات مخرجات لها أسماء ، ومرتبة بطريقة واضحة أكثر من المخرجات التى سبق إنتاجها فى المثال السابق ( لاحظ أن برنامج المثال السابق لايوجد به ترحيل لعبارات ، كما أنه يتسبب فى طباعة بيانات المدخلات مع المخرجات المحسوبة ) .

نفترض أن هذا البرنامج سوف ينفذ فى بيئة غير متداخلة . وعلى هذا ... فإننا نستمر فى طباعة بيانات المدخلات مع المخرجات المحسوبة ، وذلك لتعريف كل مشكلة ( فى ما إذا كان البرنامج ينفذ بعدة فئات بيانات مدخلات مختلفة ) .

```
PROGRAM pizzas(input,output);

(* This program calculates the selling price of a pizza,
      given the diameter and the number of extra items. *)

(* Non-interactive version *)

CONST pi = 3.14159;
      fixedcost = 0.75;
      basecost = 0.01;
      extracost = 0.0025;
VAR n : integer;
    d,area,cost,price : real;

BEGIN   (* action statements *)
   readln(d,n);
   area := pi*sqr(d)/4;
   cost := fixedcost + basecost*area + n*extracost*area;
   price := 1.5*cost;
   writeln(' Pizza size (diameter):', d:4:0,' inches');
   writeln(' Number of extra ingredients:', n:2);
   writeln;
   writeln(' The selling price is: $', price:5:2)
END.
```

فى بيئة تشغيل متداخل ، عادة ماتظهر بيانات المدخلات فى شاشة النهاية الطرفية عند إدخالها ( أثناء تنفيذ البرنامج ) . وعلى هذا ... لاتحتاج بيانات المدخلات أن تطبع مرة أخرى . وعلى أية حال ... قد لايعرف المستفيد كيف يدخل بيانات المدخلات أثناء تنفيذ البرنامج ( أى تحديد أى عناصر البيانات هو المطلوب ؟ ومتى تم إدخالها ؟ وبأى ترتيب ؟ ) . وعلى هذا ... فإن البرنامج المكتوب بصورة جيدة ليعمل فى وسط تشغيل متداخل يجب أن تنتج عنه ملقنات prompts ( أى يسأل عن بيانات مدخلات ) فى الأوقات المناسبة أثناء تنفيذ البرنامج . ويمكن تحقيق ذلك بإدخال عدد من عبارات write ( أو عبارات writeln ) داخل البرنامج ، كما هو موضح فى المثال التالى .

## مـثال (٥ -٤)

دعنا نعتبر الآن صيغة لتشغيل متداخل لصيغة البرنامج الموجودة فى المثال السابق . سوف ندخل بصفة خاصة مجموعة ملقنات ، بحيث يعرف المستفيد كيف يقوم بادخال بيانات المدخلات أثناء تنفيذ البرنامج . كما أننا لن نكتب على ذلك بيانات المدخلات مع المخرجات المحسوبة ، حيث إن هذا ليس ضروريا فى التشغيل المتداخل .

منطق البرنامجين الموجودين فى المثالين السابقين مباشر جدا . وعلى هذا ... فلاحاجة لنا بأن نهتم بطرق بديلة لتسلسل العبارات . وعلى أية حال ... فهناك بعض المعالم المرغوب فى وجودها ، والتى يمكن أن تضاف للبرنامج . فقد

```
PROGRAM pizzas(input,output);

(* This program calculates the selling price of a pizza,
      given the diameter and the number of extra items. *)

(* Interactive version *)

CONST pi = 3.14159;
      fixedcost = 0.75;
      basecost = 0.01;
      extracost = 0.0025;
VAR n : integer;
    d,area,cost,price : real;
```

```
BEGIN   (* action statements *)
   page;
   write(' Enter the pizza size (diameter), in inches: ');
   readln(d);
   write(' Enter the number of extra ingredients: ');
   readln(n);
   area := pi*sqr(d)/4;
   cost := fixedcost + basecost*area + n*extracost*area;
   price := 1.5*cost;
   writeln;
   writeln(' The selling price is: $', price:5:2)
END.
```

نختار – على سبيل المثال – أن نكتب التكاليف المختلفة ( أى قيم الثوابت ) مع النتائج المحسوبة . قد تكون هذه المعلومات مفيدة إذا نفذ البرنامج العديد من المرات باستخدام بيانات تكاليف مختلفة فى كل مرة من مرات التنفيذ .

وإحدى السمات التى قد تكون مفيدة ، هى إمكانية تنفيذ البرنامج على التوالى لعدة فئات مختلفة من بيانات المدخلات . وسوف نرى كيف يمكن تحقيق ذلك فى الفصل القادم .

## ٣ – إدخال البرنامج داخل الكمبيوتر : 3. ENTERING THE PROGRAM INTO THE COMPUTER

بعد كتابة البرنامج ، يجب إدخاله داخل الكمبيوتر قبل أن يمكن أن يترجم وينفذ . وعادة مايتحقق ذلك بإحدى طريقتين ممكنتين ، والطريقة الأكثر شيوعا هى استخدام منقح editor .

وعمليا تحتوى كل نظم الكمبيوتر الحديثة على بعض أنواع المنقحات التى يمكن استخدامها فى إدخال ملف نصوص text file ( ويمكن أن يكون ملف النصوص عبارة عن برنامج أو توثيق نص ، مثل الخطاب ، أو ملف البيانات وغيرها ) . وبعض المنقحات تكون موجهة للسطر line - oriented ، بينما يكون بعضها موجها للرمز Character - orienter . والمنقحات الموجهة للرمز ، والتى تستخدم مع وحدات العرض المرئى المتداخل عادة مايشار إليها بأنها منقحات شاشة Screen editors . ومثل هذه المنقحات مريحة عمليا فى استخدامها ، حيث إنها تسمح بعرض أجزاء كبيرة من الملف على الشاشة فى أى وقت . ويمكن تحريك نقطة البداية التى تضئ ضوءا متقطعا لأى موقع فى الشاشة ، وهذا يسهل من الادخال والتعديل والحذف فى النص .

وهناك منقحات مختلفة شائعة الاستخدام لكل أنواع الكمبيوتر . ويمكن استخدام أى منقح فى إدخال برنامج البسكال وملفات البيانات المصاحبة له ، بالرغم من أن بعض نظم البسكال تحتوى على منقحات خاصة بها . وبغض النظر عن نوع المنقح المستخدم ، فإن عملية إدخال البرنامج تكون عبارة عن كتابة برنامج البسكال داخل الكمبيوتر ، وذلك سطرا سطرا . ( ويمكن تحقيق ذلك باستخدام كل من المنقحات الموجهة للسطر والموجهة للرمز ) . وبعد ذلك يحدد اسم ملف للبرنامج ، ويخزن البرنامج فى ذاكرة الكمبيوتر أو عن طريق إحدى وحدات التخزين المساعد . وفى بعض الأحيان تضاف لاحقة suffix ، مثل PAS إلى اسم الملف ، وذلك لتعريف الملف بأنه برنامج بسكال . ( وعادة ماتسمى مثل هذه اللاحقة بأنها اتساع extension ) . ويظل البرنامج فى هذه الصورة ، حتى يكون معدا لترجمته بمترجم البسكال .

والطريقة البديلة لاستخدام المنقح هى قراءة برنامج البسكال فى الكمبيوتر من مجموعة من البطاقات المثقبة . وفى واقع الأمر ... فإن هذه الطريقة أصبحت متقادمة فى إدخال البرامج ، بالرغم من أنها مازالت تستخدم مع بعض أجهزة الكمبيوتر الكبيرة . واستخدام البطاقات المثقبة أقل راحة من استخدام المنقح ، إلا أنها أفضل من لاشئ .

تستخدم العديد من نظم الكمبيوتر الكبيرة كل من تشغيل الدفعة وتشغيل المشاركة الزمنية مع أحد صيغ التنقيح فى وسط الخط المفتوح . ويسمح هذا للمستفيد بأن يختار طريقة العمل الأكثر راحة له . فمثلا يمكن إدخال برنامج جديد والبيانات المصاحبة له بواسطة منقح ، إلا أن النتائج المحسوبة ( أى محتويات ماف المخرجات ) يمكن أن تخرج عن طريق طابع أسطر .

ويجب أن يحدد القارئ نوع عملية إدخال البرنامج المتاحة له . فإذا ماكان هناك منقح متاح له ، فيجب عليه أن يحصل على دليل التشغيل ، أو على أى مجموعة تعليمات تصف له بالضبط كيفية استخدام المنقح .

## ٤ – ترجمة وتنفيذ البرنامج : 4. COMPILING AND EXECUTING THE PROGRAM

بعد الانتهاء من إدخال البرنامج الكامل بطريقة صحيحة داخل الكمبيوتر ، يمكن ترجمته وتنفيذه . وعادة ماتتحقق عملية الترجمة تلقائيا كاستجابة لأمر معين ( مثل COMPILE ) . وفى بعض الأحيان يكون من الضرورى أيضا توصيل link برنامج التشغيل المترجم إلى إجراء routines واحد أو أكثر من برامج المكتبة ، وذلك لتنفيذ البرنامج . ويتحقق ذلك تلقائيا فى معظم الحالات أيضا وذلك بإصدار أمر بسيط ( مثل LINK ) .

سوف ينتج عن الترجمة والاتصال الناجحين لبرنامج البسكال برنامج تشغيل ( بلغة الآلة ) قابل للتنفيذ ، يمكن أن ينفذ بعد ذلك إذا ماصدر أحد أوامر النظام المناسبة ( مثل EXECUTE ) . وسوف تختلف تفاصيل التنفيذ على أية حال طبقا لنوع البيئة المستخدمة فى التشغيل ... ففى بيئة غير متداخلة – على سبيل المثال – يكون مطلوبا ملف بيانات مدخلات منفصل . ويجب إدخال هذا الملف داخل الكمبيوتر قبل أن يمكن تنفيذ برنامج التشغيل . وعند تنفيذ البرنامج تقرأ بيانات المدخلات تلقائيا من ملف البيانات ، ويتم تشغيلها لينتج عنها مجموعة من بيانات المخرجات ، والتى تخزن تلقائيا فى ملف بيانات المخرجات . ويجب أن يطبع ملف المخرجات هذا ، أو يعرض على الشاشة كعملية منفصلة بعد الانتهاء من تنفيذ البرنامج .

### مثال (٥ – ٥)

افرض أننا نرغب فى تحديد سعر بيع الفطائر التى قطرها 10 بوصة ، بها ثلاثة مكونات إضافية . سوف نستخدم برنامج البسكال لبيئة غير متداخلة ، والموجود فى المثال رقم ٥ – ٣ .

ندخل البرنامج فى الكمبيوتر أولا ، وذلك عن طريق نهاية طرفية باستخدام منقح نصوص . ويجب أن نكون حذرين عند الإدخال ، وذلك بتصحيح كل الأخطاء . ويجب إعطاء اسم لتمييز البرنامج ، مثل .PIZZAS.PAS ، بعد ذلك ندخل ملف بيانات المدخلات ، والذى يحتوى على القيمتين التاليتين :

```
10.0    3
```

وسوف يخزن هذا الملف ككينونة مستقلة تحت اسم :

```
INPUT.DAT.
```

والآن نكون مستعدين لترجمة وتوصيل البرنامج . ويمكن عمل ذلك بإدخال الأمر التالى :

```
COMPILE PIZZAS.PAS
```

عن طريق النهاية الطرفية . ( ويتم تنشيط الاتصال تلقائيا إذا ماتمت ترجمة البرنامج بنجاح ) . وعند ذلك يكون لدينا ملف جديد اسمه .PIZZAS.OBJ ، والذى يحتوى على برنامج تشغيل مترجم . هذا يكون بالإضافة بالطبع إلى الملف الأساسى .PIZZAS.PAS الذى يحتوى على برنامج المصدر بلغة البسكال .

ولتنفيذ برنامج التشغيل ، فإننا نكتب الأمر

```
EXECUTE PIZZAS
```

عن طريق النهاية الطرفية . ( لاتكون هناك حاجة لكتابة العنوان OBJ. ) وعلى هذا ... تنفذ التعليمات الموجودة فى برنامج البسكال الأساسى ، وذلك بالرغم من أن برنامج التشغيل - وليس برنامج المصدر الأساسى - هـو الذى ينـفذ . وعلى هذا ... تحدث الأشياء التالية :

(١) تقرأ قيم n , d من ملف المدخلات .

(٢) تحسب المساحة والتكلفة الكلية والسعر .

(٣) تكتب قيم price , n , d فى ملف المخرجات .

وهذا ينهى جزء التنفيذ للبرنامج . ولكى نرى النتائج المحسوبة على أية حال ، يكون من اللازم عرض ملف المخرجات . وعلى هذا ... فإننا نصدر الأمر :

```
TYPE OUTPUT.DAT
```

عن طريق النهاية الطرفية . وسوف يتسبب ذلك فى عرض المخرجات التالية :

```
Pizza size (diameter): 10. inches
Number of extra ingredients: 3

The selling price is: $ 3.19
```

وينفذ برنامج البسكال فى التشغيل المتداخل بطريقة مختلفة . ولايكون مطلوب ملفات بيانات مدخلات ، أو ملفات بيانات مخرجات بصفة خاصة . وبدلا من ذلك يتم إدخال بيانات المدخلات فى الكمبيوتر أثناء تنفيذ البرنامج . وبالمثل تعرض بيانات المخرجات عند إنتاجها أثناء تنفيذ البرنامج أيضا .وبالنسبة لمعظم التطبيقات ، فإن هذه الطريقة للتشغيل تكون أكثر راحة .

## مـثال (٥ -٦)

دعنا نعيد المشكلة الموجودة فى مثال ٥ - ٥ ، باستخدام برنامج بسكال فى التشغيل المتداخل ، الموجود فى مثال ٥ - ٤ سوف ينفذ البرنامج على جهاز ميكرو كمبيوتر ، مع استخدام نظم برامج للنظام تحتوى على منقح ، وعلى مترجم بسـكال .

نكتب مرة أخرى البرنامج فى الكمبيوتر باستخدام منقح النصوص ، مع تصحيح أى أخطاء تحدث فى إدخال البرنامج . عند ذلك يخزن البرنامج كملف نصوص تحت اسم PIZZAS.PAS. . لانحتاج الآن على أية حال ملفا للمدخلات . وبدلا من ذلك ... فإننا نستمر فى ترجمة وتوصيل البرنامج بإدخال الأوامر :

```
COMPILE PIZZAS.PAS
LINK PIZZAS
```

فإذا مانفذ الأمران LINK , COMPILE بنجاح ؛ فسوف ينتج عن ذلك برنامج تشغيل (PIZZAS.OBJ) ، وبعد ذلك نستطيع أن نستمر فى التنفيذ .. وإلا فإنه يكون من الضرورى العودة إلى برنامج المصدر (PIZZAS.PAS) ، وإجراء أى تعديلات ضرورية ( عن طريق المنقح ) ثم نبدأ العملية بنفس التسلسل مرة أخرى . ( هذه العملية المتكررة يجب أن تتبع بالطبع فى كل من صيغ بسكال المتداخلة وغير المتداخلة ) .

دعنا نفترض الآن أن الترجمة والتوصيل حدثا بنجاح . ولتنفيذ البرنامج ، فإننا نكتب :

```
EXECUTE PIZZAS
```

ويبدأ عند ذلك تنفيذ التعليمات الموجودة فى برنامج البسكال الأصلى . وسوف يلقن الكمبيوتر بصفة خاصة الحاجة إلى حجم الفطيرة ، وذلك بإنتاج الرسالة التالية على الشاشة :

```
Enter the pizza size (diameter), in inches:
```

يستجيب المستفيد على ذلك بإدخال القيمة 10.0 ، ويسأل الكمبيوتر بعد ذلك عن عدد العناصر الإضافية بإنتاجه الرسالة التالية :

```
Enter the number of extra ingredients:
```

ويدخل المستفيد عند ذلك القيمة 3 .

وكل بيانات المدخلات المطلوبة تم إدخالها الآن . وعلى هذا ... تحسب المساحة والتكلفة الكلية والسعر مع عرض السعر على الشاشة فى الصورة التالية :

```
The selling price is: $ 3.19
```

لاحظ أن النتيجة النهائية ظهرت مباشرة كما حسبت ، فليس هناك حاجة لملف بيانات المخرجات .

وفيما يلى ملخصا لجزء التداخل ، مع وضع خط تحت مايرد به المستفيد ، وذلك بغرض التوضيح :

```
(program statements . . . entered via an editor)

COMPILE PIZZAS.PAS

LINK PIZZAS

EXECUTE PIZZAS

Enter the pizza size (diameter), in inches: 10.0
Enter the number of extra ingredients: 3

The selling price is: $ 3.19
```

الاجراءات الموضحة فى المثالين السابقين تمثل الإجراءات شائعة الاستخدام ، وذلك بالرغم من إمكانية وجود بعض الاختلافات من جهاز الكمبيوتر لجهاز آخر . فيوجد على سبيل المثال بعض نظم بسكال تستخدم أمرا معينا ( مثل RUN ) يتسبب فى ترجمة برنامج المصدر المكتوب بالبسكال ، مع توصيله وتنفيذه على التوالى . ويجب على القارئ أن يحدد الأوامر الخاصة بالترجمة والتوصيل والتنفيذ المتاحة مع الجهاز المتاح له .

وفى النهاية يجب أن يتذكر القارئ أن الترجمة والتوصيل والتنفيذ الناجحة لبرنامج البسكال عادة ماتتطلب محاولات متعددة بسبب الأخطاء التى تحدث فى معظم البرامج الجديدة . وهذا صحيح بالنسبة للبرامج التى يكتبها كل من المبرمجين ذوى الخبرة والمبتدئين . وعلى هذا ... فقد تبدو الإجراءات مرهقة ، وذلك بالنسبة للمبتدئين بصفة خاصة . ويجب على أية حال فهم أن تكرار حدوث مثل هذه الأخطاء سوف يقل مع اكتساب المبرمج الخبرة . وعلى هذا ... فيجب أن يكون المبتدئون حذرين من ناحية أنه قد يحدث لهم إحباط مع أول عدة برامج ، فى نفس الوقت الذى تتحسن فيه مقدرتهم .

## ٥ - تشخيص الخطأ : 5. ERROR DIAGNOSTICS

عادة ماتظل أخطاء البرمجة غير مكتشفة ، حتى تحدث محاولة ترجمة البرنامج . وبمجرد إصدار الترجمة COMPILE ، تظهر الأخطاء ، حيث إنها تمنع من ترجمة البرنامج بنجاح . وبعض هذه الأخطاء شائعة الاستخدام عادة ماتوضع على أنها ثوابت ومتغيرات غير صحيحة ، أو إشارة إلى متغير غير موضح أو استخدام تنقيط خاطئ . يشار إلى مثل هذه الأخطاء بأنها أخطاء تكوينية syntactical errors ، أو أخطاء فى القواعد grammatical errors .

وسوف تنتج معظم صيغ البسكال رسائل تشخيصية عند اكتشاف الأخطاء التكوينية . وعادة مالاتكون هذه الأخطاء التكوينية مباشرة فى معناها ، كما أنها لاتساعد على تعريف طبيعة وموقع الخطأ .

### مـثال (٥ -٧)

فيما يلى صيغة متداخلة أخرى من صيغ برامج البسكال الموجود فى مثال ٥ - ٤ ، والذى يحسب سعر بيع الفطيرة إذا ماأعطى حجم الفطيرة ( القطر ) ، وعدد المكونات الإضافية . ويختلف هذا البرنامج عن الصيغة السابقة فى أنه يحتوى على العديد من الأخطاء التكوينية .

وعند محاولة ترجمة البرنامج نحصل على قائمة الأخطاء التالية :

```
PROGRAM pizzas(input,output);

(* This program calculates the selling price of a pizza,
      given the diameter and the number of extra items.

(* Interactive version *)

     CONST pi = 3.14159
           fixedcost = 0.75;
                      ^
****** ERROR 346 MISSING ;
           basecost := 0.01;
                     ^
****** WARNING 149 := ASSUMED =
           extracost = 0.0025;
     VAR n : integer;
         d,area,cost,price = real;
                          ^
****** WARNING 148 = ASSUMED :

     BEGIN   (* action statements *)
        page;
        write(' Enter the pizza size (diameter), in inches: );
                                                             ^
****** WARNING 105 END OF STRING NOT FOUND

****** WARNING 169 INSERT )
        readln(d);
        ^
****** ERROR 185 INVALID SYMBOL BEGIN SKIP
        ^
****** WARNING 164 INSERT ;
        ^
****** ERROR 186 END SKIP
```

(تكملة البرنامج فى الصفحة التالية)

```
****** ERROR 185 INVALID SYMBOL BEGIN SKIP
                     ^
****** ERROR 186 END SKIP
        write(' Enter the number of extra ingredients: ');
        readln(n);
        area = pi*sqr(d)/4;
             ^
****** WARNING 150 = ASSUMED :=
        cost := fixedcost + basecost*area + n*extracost*area
                                                            ^
****** WARNING 164 INSERT ;
        price := 1.5cost;
                    ^
****** ERROR 185 INVALID SYMBOL BEGIN SKIP
                    ^
****** ERROR 186 END SKIP
        writeln;
        writeln(' The selling price is: $', price,5,2)
     END

****** ERROR 146 UNEXPECTED END OF FILE

****** FATAL PROGRAM ERRORS - COMPILER CANNOT CONTINUE!
```

ويجب ذكر عدة نقاط . أولا : لاحظ أنه عادة ماتكون رسائل الخطأ خفية ، وبعضها يكون من الصعب فهمه . ثانيا : لاحظ أن العديد من هذه الأخطاء يتم تصحيحها تلقائيا . وتعرف هذه التصحيحات بأنها رسائل تحذيرية WARNING ، بدلا من أنها رسائل خطأ ERROR . وأخيرا لاحظ أن عدم وجود ( * فى نهاية أول تعليق لم تكتشف x) ( وقد تم تفسير التعليقين كتعليق واحد طويل يحتوى على أقواس داخلية ) .

وسوف تتغير رسائل الخطأ والرسائل التحذيرية من صيغة لأخرى من صيغ البسكال . وهذه الأخطاء تمثل الأخطاء المعتادة فى برامج البسكال .

وعادة ماتكون الأخطاء التكوينية واضحة جدا ، نظرا للأعراض التى تنتجها . أما الأخطاء المنطقية ، فتكون غير واضحة . وهنا يحمل البرنامج تعليمات البرنامج بطريقة صحيحة ، وهى خالية من الأخطاء التكوينية ، إلا أن هناك تعليمات خاطئة منطقيا .

وفى بعض الأحيان ينتج عن الخطأ المنطقى شرط يمكن للكمبيوتر أن يميزه . وقد ينتج مثل هذا الموقف من ظهور قيم عددية كبيرة ( تتعدى أقصى عدد مسموح بتخزينه فى الكمبيوتر ) ، أو من محاولة حساب الجذر التربيعى لعدد سالب ، وما إلى ذلك من أخطاء . وعادة ماتظهر رسائل تشخيصية فى مثل هذه المواقف لتسهيل تعريف الأخطاء وتصحيحها . وعادة ماتسمى هذه التشخصيات بأنها تشخصيات التنفيذ EXECUTION ، وذلك لتمييزها عن تشخصيات الترجمة COMPILATION التى سبق ذكرها .

## مـــثال (٥ -٨)

*الجذور الحقيقية لمعادلة من الدرجة الثانية* : افرض أنه مطلوب حساب الجذور الحقيقية للمعادلة من الدرجة الثانية التالية :

$$ax^2 + bx + c = 0$$

باستخدام الصيغة التالية :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

وفيما يلى برنامج بسكال يجرى هذه الحسابات :

```
PROGRAM realroots(input,output);

(* This program calculates the real roots of a quadratic equation *)

VAR a,b,c,d,x1,x2 : real;

BEGIN   (* action statements *)
   readln(a,b,c);
   d := sqr(b)-4*a*c;
   x1 := (-b+sqrt(d))/(2*a);
   x2 := (-b-sqrt(d))/(2*a);
   writeln(' a=',a,'   b=',b,'   c=',c);
   writeln;
   writeln(' x1=',x1,'   x2=',x2)
END.
```

هذا البرنامج لايحتوى على أى أخطاء تكوينية على الإطلاق ، إلا أنه لايستطيع أن يؤدى الحسابات لقيم سالبة لـ :

```
sqr(b) - 4 a*c.
```

والأكثر من هذا ... قد تظهر صعوبات عددية إذا ماكانت هناك قيمة صغيرة جداً أو قيمة كبيرة جدا للمتغير ، أو إذا ماكانت a = 0 . وفى كل من هذه الحالات يظهر تشخيص تنفيذ .

افرض على سبيل المثال أن المدخلات التالية استخدمت عند تشغيل البرنامج :

```
a=1.0      b=2.0      c=3.0
```

تحدث الترجمة دون أى صعوبة . وعند تنفيذ برنامج التشغيل تظهر رسالة الخطأ التالية :

```
? Error: SQRT of Negative Argument
  Error Code 2104
```

وعند ذلك ينتهى كل شئ ، حيث إن تنفيذ البرنامج لايمكن أن يستمر بعد الوصول الى هذه النقطة .

وبالمثل افرض أن المدخلات التالية استخدمت عند تشغيل البرنامج :

```
a=1E-30    b=1E+10    c=1E+36
```

عند ذلك ينتج النظام رسالة الخطأ التالية :

```
? Error: REAL Math Overflow
  Error Code 2101
```

وذلك عند محاولة تنفيذ البرنامج .

## ٦ – تصحيح المنطق : 6. LOGICAL DEBUGGING

سبق أن رأينا أن الأخطاء التكوينية ، وبعض الأخطاء المنطقية تتسبب فى إظهار رسائل تشخيصية عند ترجمة البرنامج أو تنفيذه . ومن السهل اكتشاف مثل هذه الأخطاء وتصحيحها ، إلا أن بعض أنواع الأخطاء المنطقية يكون من الصعب جدا اكتشافها ، حيث إنه فى مثل هذه الحالات تبدو مخرجات البرنامج كما لو كانت خالية من الأخطاء المنطقية ، حتى عند معرفة أنها موجودة ( كما فى حالة ظهور مخرجات تبدو أنها صحيحة ) . وعلى هذا ... تكون هناك حاجة إلى اكتشاف مثل هذه الأخطاء وتصحيحها . والعمل الذى يؤدى لتحقيق هذا الغرض يسمى بتصحيح المنطق .

### اكتشاف الأخطاء : Detecting Errors

إن أول خطوة فى مهاجمة الأخطاء المنطقية هو تحديد مواقع تواجدها . ويمكن تحقيق ذلك فى بعض الأحيان باختبار البرنامج الجديد ببيانات اختبارية ، فإذا لم يتم الحصول على نفس النتائج المرجوة ، فهذا يؤكد أن البرنامج به أخطاء . حتى إذا ماتم الحصول على نتائج صحيحة ، فلانستطيع التأكد تماما بأن البرنامج خالٍ من الأخطاء ، حيث إن بعض الأخطاء تتسبب فى ظهور نتائج خاطئة تحت ظروف معينة فقط . ( كما فى حالة قيم معينة لبيانات المدخلات ، أو فى حالة تنفيذ اختيارات معينة للبرنامج ) . وعلى هذا ... يجب أن يمر البرنامج الجديد خلال اختبار دقيق قبل أن يعتبر صحيحا . وهذا صحيح بصفة خاصة بالنسبة للبرامج المعقدة ، والبرامج التى سوف يستخدمها العديد من الناس .

وكقاعدة عامة ... تجرى الحسابات يدويا بمساعدة حاسبة جيب ، بهدف معرفة الإجابة . وبالنسبة لبعض المشاكل – على أية حال – فإن كمية العمل اللازمة لأداء مثل هذه الحسابات تكون كبيرة جدا ، وتمنع من إجراء مثل هـذا الاختبار . ( تذكر أن الحسابات التى تستغرق عدة دقائق لتنفيذها بواسطة الكمبيوتر قد تتطلب عدة أسابيع لحلها يدويا ) . وعلى هذا ... فإن حساب العينة لايمكن استخدامه بصفة دائمة لاختبار البرنامج الجديد . وتصحيح منطق مثل هذه البرامج يمكن أن يكون صعبا بصفة خاصة ، بالرغم من أن المبرمج عادة مايمكنه اكتشاف وجود أخطاء منطقية عن طريق دراسة النتائج التى يتم الحصول عليها من البرنامج دراسة دقيقة لمعرفة ما إذا كانت مناسبة أم لا .

### تصحيح الأخطاء : Correcting Errors

بعد معرفة أن البرنامج يحتوى على أخطاء منطقية ، تكون هناك حاجة إلى إيجاد هذه الأخطاء . واكتشاف الأخطاء عادة مايبدأ بمراجعة دقيقة لكل مجموعة من مجموعات العبارات المنطقية داخل البرنامج . وبمعرفة أنه هناك خطأ معين فى مكان ما ، فعادة مايستطيع المبرمج أن يحدد موقع الخطأ عن طريق هذه الدراسة الدقيقة . فإذا لم يستطع أن يجد الخطأ ، فترك البرنامج لفترة معينة يساعد فى بعض الأحيان ( وهذا صحيح بصفة خاصة إذا ماكان المبرمج مرهقا من العمل ) . وليس من غير المعتاد للمبرمج أن يمر على الخطأ دون اكتشافه من أول مرة .

وإذا لم يمكن تحديد موقع الخطأ عن طريق فحص البرنامج ، فيجب تعديل البرنامج ليطبع نتائج مرحلية معينة ، ثم يستمر فى التنفيذ . ( مثل هذا الأسلوب يشار إليه فى بعض الأحيان بانه تتبع TRACING ) . وعادة مايصبح مصدر الخطأ واضحا عند فحص هذه النتائج المرحلية فحصا دقيقا . وعادة مايستطيع المبرمج تعريف احدى مناطق البرنامج والتى يحدث داخلها الخطأ . وكلما ازداد حجم المخرجات المرحلية ، ازدادت فرصة تحديد مصدر الخطأ .

وفى بعض الأحيان لايمكن تحديد موقع الخطأ ببساطة ، بالرغم من استخدام معظم أساليب التصحيح . وفى مثل هذه المواقف يتوقع للمبرمج المبتدئ أنه يواجه مشكلة خارج حدود معرفته ، مثل أخطاء نظم المكونات ، أو أخطاء المترجم نفسه . وفى كل الحالات تقريبا تكون المشكلة فى منطق البرنامج . وعلى هذا ... يجب على المبرمج المبتدئ أن يقاوم إغراء إلقاء اللوم على الكمبيوتر ، ويفحص برنامجه بدقة ليكتشف أخطاء البرمجة . ( أخطاء نظم المكونات نادرة الحدوث ، كما أن أخطاء المترجم لاتظهر عادة إلا عند استخدام مترجم جديد ، لكن يتم تصحيحها خلال فترة بسيطة من استخدام المترجم ) .

وأخيرا يجب أن يميز القارئ حقيقة أن بعض الأخطاء المنطقية غير قابلة للهرب فى برمجة الكمبيوتر ، وعلى هـذا ... يجب على المبرمج الواعى أن يبذل كل المحاولات لتقليل حدوث هذه الأخطاء ، وعلى هذا ... يجب على المبرمج أن يتوقع أن هناك حاجة إلى تصحيح منطقى عند كتابته برامج بسكال واقعية وذات معنى .

## مـثال (٥-٩)

تقويم كثيرة حدود : لقد كتب أحد الطلبة برنامج بسكال لتقويم كثيرة حدود لها الشكل التالى :

$$y = \left(\frac{x-1}{x}\right) + \frac{1}{2}\left(\frac{x-1}{x}\right)^2 + \frac{1}{3}\left(\frac{x-1}{x}\right)^3 + \frac{1}{4}\left(\frac{x-1}{x}\right)^4 + \frac{1}{5}\left(\frac{x-1}{x}\right)^5$$

ولكى يبسط الطالب البرمجة ، فقد قام بتعريف متغير جديد u على أنه :

$$u = \left(\frac{x-1}{x}\right)$$

وبذلك تحولت الصيغة إلى مايلى :

$$y = u + \frac{u^2}{2} + \frac{u^3}{3} + \frac{u^4}{4} + \frac{u^5}{5}$$

وفيما يلى البرنامج الكامل الذى أعده الطالب :

```
PROGRAM formula(input,output);

(* Program to evaluate an algebraic formula *)

VAR u,x,y : real;
BEGIN
   readln(x);
   u := x-1/x;
   y := u+sqr(u/2)+(u/3)*sqr(u/3)+sqr(u/4)*sqr(u/4)+(u/5)*sqr(u/5)*sqr(u/5);
   writeln(' x=',x,'      y=',y)
END.
```

ويعرف الطالب أن قيمة y يجب أن تكون حوالى 0.69 عندما تكون x = 2 ، إلا أن مخرجات البرنامج كانت على النحو التالى :

```
x= 2.0000000E+00        y= 2.2097050E+00
```

وقد استنتج الطالب – على ذلك – أن البرنامج يحتوى على أخطاء منطقية ، والتى يجب أن يعرفها ويصححها .

وبعد الفحص الدقيق للبرنامج تحقق الطالب من أن أول عبارة تحديد غير صحيحة . ويجب أن تكون على الصورة التالية :

```
u := (x-1)/x;
```

وقد قام الطالب بتصحيح البرنامج ، وأعاد تنفيذه ، مستخدما x = 2 ؛ وحصل على المخرجات التالية :

```
x= 2.0000000E+00        y= 5.6738380E-01
```

وهذا يبين أنه مازال هناك خطأ منطقى .

وبعد دراسة إضافية أخرى للبرنامج ، اكتشف أن عبارة التحديد الثانية خاطئة أيضا . ويجب أن تكون على الصورة التالية :

```
y := u+sqr(u)/2+(u*sqr(u))/3+(sqr(u)*sqr(u))/4+(u*sqr(u)*sqr(u))/5;
```

فقام بتعديل البرنامج وإعادة تنفيذه ، ليحصل على النتيجة الصحيحة التالية :

```
x= 2.0000000E+00        y= 6.8854160E-01
```

## أسئلة للمراجعة : Review Questions

(١) ماذا تعنى البرمجة من القمة إلى القاعدة ؟ وماهى مميزاتها ؟ وكيف يمكن تنفيذها ؟

(٢) ماهى الشفرة الشبيهة ؟

(٣) ماذا تعنى البرمجة من القاعدة إلى القمة ؟ وماهو الفرق بينها وبين البرمجة من القمة إلى القاعدة ؟

(٤) لخص التكوين الشامل لبرنامج البسكال .

(٥) كيف تستخدم الفاصلة المنقوطة فى البسكال ؟ وفى أى مكان يجب ظهورها ؟

(٦) ماهى علامة التنقيط الخاصة التى يجب أن تظهر فى نهاية كل برنامج من برامج البسكال ؟

(٧) ماهو الغرض من BEGIN , END ؟ وماهو التنقيط الذى يصاحب هاتين الكلمتين الأساسيتين ؟

(٨) ماهى درجة المرونة المتاحة للمبرمج فى تسلسل العبارات المنطقى داخل برنامج البسكال ؟ وضح إجابتك .

(٩) لماذا ترحل بعض العبارات داخل برنامج البسكال ؟ وهل هذا الترحيل ضرورى ؟

(١٠) ماهى أسباب وضع التعليقات فى برنامج البسكال ؟ وماهى الكثافة التى يجب أن تظهر بها هذه التعليقات ؟

(١١) اذكر عاملين يسهمان فى إنتاج بيانات مخرجات واضحة .

(١٢) وضح كيف تؤثر بيئة البرمجة ( بيئة متداخلة وبيئة غير متداخلة ) على المعالم الخاصة الموجودة فى البرنامج للمدخلات والمخرجات .

(١٣) ماهو التلقين ؟ تحت أى ظروف يجب أن يظهر التلقين فى برنامج البسكال ؟

(١٤) ماهو ملف النصوص ؟

(١٥) ماهو الفرق بين المنقح الموجه للسطر ، والمنقح الموجه للرمز ؟

(١٦) اذكر طريقتين مختلفتين لإدخال برنامج البسكال فى الكمبيوتر . أى هذه الطرق تفضلها ؟

(١٧) ماهو الفرق بين ترجمة برنامج البسكال وتنفيذه ؟

(١٨) مامعنى ربط linking ؟ وكيف يختلف الربط عن الترجمة ؟

(١٩) اذكر بعض الاختلافات الأساسية فى طرق تنفيذ برنامج البسكال طبقا لما إذا كان وسط الحسابات متداخلاً أم لا .

(٢٠) ماذا يعنى الخطأ التكوينى ؟

(٢١) ماذا يعنى الخطأ المنطقى ؟ كيف تختلف الأخطاء التكوينية عن الأخطاء المنطقية ؟

(٢٢) اذكر بعض الأخطاء التكوينية الشائعة .

(٢٣) اذكر بعض الأخطاء المنطقية الشائعة .

(٢٤) ماهى الرسائل التشخيصية ؟

(٢٥) ماهو الفرق بين تشخيصات الترجمة وتشخيصات التنفيذ ؟ اذكر بعض المواقف التى تظهر فيها رسائل تشخيصية .

(٢٦) مامعنى تصحيح المنطق ؟ اذكر بعض إجراءات التصحيح الشائعة .

(٢٧) مامعنى التتبع tracing ؟ وكيف يكون مفيدا ؟

## مشـــاكـــل : Problems

التمارين التالية تهتم بجمع المعلومات أكثر من اهتمامها بحل المشاكل .

(٢٨) إذا ماكانت المشاركة الزمنية مستخدمة فى كليتك أو فى مكتبك ، فاحصل على إجابة للأسئلة التالية :

أ ) هل توجد نهايات طرفية للعرض المرئى ؟ هل توجد نهايات طرفية لطباعة نسخ دائمة ؟

ب ) كيف يمكن بدء وإنهاء عمل النهاية الطرفية ؟

ج ) كيف يمكن حذف رمز من رموز أحد الأسطر المكتوبة قبل إرسال السطر إلى الكمبيوتر ؟ وكيف يمكن حذف سطر كامل ؟

د ) كيف يمكن نقل السطر إلى الكمبيوتر ؟

هـ ) هل يمكنك الحصول على نسخة دائمة ؟ إذا كانت الإجابة نعم ، فحدد كيف يتم ذلك .

و ) هل من المطلوب إجراء اتصال هاتفى لعمل اتصال مع الكمبيوتر ؟ إذا كانت الإجابة نعم ، فما هى الإجراءات التى تتبع ؟

ز ) حدد بالضبط كيف تبدأ العمل مع الكمبيوتر الكبير وكيف تنهيه .

ح ) هل يمكن للنهاية الطرفية أن تعمل بطريقة محلية ( أى تعمل كوحدة مستقلة بذاتها منفصلة عن عمل الكمبيوتر الكبير ) ؟

ط ) ماهو المنقح أو المنقحات المتاحة فى النظام ؟ وكيف تجرى وظائف التنقيح المعتادة ( مثل الحذف والإضافة وغيرها ) ؟

ى ) كيف يمكن الاتصال بمترجم البسكال فى النظام المتاح لك ؟ وماهى الإجراءات اللازمة لترجمة البرنامج ولتوصيله وتنفيذه ؟

ك ) ماهى تكلفة استخدام الكمبيوتر ؟

(٢٩) إذا ماكان تشغيل الدفعة مستخدما فى كليتك أو فى مكتبك ، فاحصل على إجابة للأسئلة التالية :

أ ) هل توجد آلات تثقيب بطاقات ؟ إذا كانت الأجابة نعم ، فكيف يتم فتحها وإغلاقها ؟ كيف تتم تغذية البطاقات خلال الآلة ؟ كيف تثقب البطاقة ؟ كيف يعاد إنتاج بطاقة أخرى ؟

ب ) ماهو موقع قارئ البطاقات ؟ كيف يمكن قراءة مجموعة من البطاقات المثقبة بواسطة قارئ البطاقات ؟

ج ) ماهى بطاقات التحكم المطلوبة ؟

د ) كيف يمكن الاتصال بمترجم البسكال فى النظام المتاح لك ؟ وماهى الإجراءات اللازمة لترجمة البرنامج ولتوصيله وتنفيذه ؟

هـ ) كيف يمكن تخزين البرنامج أو ملف البيانات فى هذا النظام ؟ وكيف يمكن الاتصال بالبرنامج أو بملف البيانات ؟ وكيف يمكن حذف أى منهما ؟

و ) حدد موقع طابع الأسطر . ماهو شكل المخرجات المطبوعة ؟

ز ) ماهى تكلفة استخدام الكمبيوتر ؟

(٣٠) إذا ماكان يستخدم جهاز ميكروكمبيوتر فى كليتك أو فى مكتبك ، فاحصل على إجابة للأسئلة التالية :

أ ) حدد بالضبط المعدات المتاحة لك ( طابعات ، أو وحدات ذاكرة مساعدة وغيرها ) .

ب ) كيف يمكن فتح وإغلاق الجهاز ؟

ج ) كيف تخزن البرامج ؟ وكيف تعرض ؟ وكيف تنقل من إحدى أنواع الذاكرات إلى ذاكرة أخرى ؟

د ) كيف يمكن حذف رمز واحد من رموز أحد الأسطر المكتوبة قبل إرسال السطر إلى الكمبيوتر ؟ وكيف يمكن حذف سطر كامل ؟ .

هـ ) كيف يمكن نقل السطر إلى الكمبيوتر ؟

و ) كيف يمكن الاتصال بالمنقح ؟ وكيف يمكن أداء وظائف التنقيح المعتادة ( مثل الحذف والإضافة وغيرها ) ؟

ز ) كيف يمكن الاتصال بمترجم البسكال فى النظام المتاح لك ؟ وماهى الإجراءات اللازمة لترجمة البرنامج ولتوصيله وتنفيذه ؟

ح ) ماهى تكلفة استخدام الكمبيوتر ؟

## مشاكل برمجة : Programming Problems

(٣١) يمثل مثال ١ – ٧ فى الفصل الأول من الكتاب برنامجا لحساب مساحة الدائرة بمعرفة نصف قطر الدائرة . قم بإدخال هذا البرنامج فى الكمبيوتر المتاح لك . تأكد من تصحيح أى أخطاء تحدث عند كتابة البرنامج . اسرد محتويات البرنامج بعد تخزينه فى الكمبيوتر . عندما تتأكد من صحة البرنامج ، ترجم البرنامج ، ثم نفذ برنامج التشغيل ، مستخدما قيما مختلفة عديدة لنصف القطر . تحقق من صحة الإجابات المحسوبة ، وذلك بمقارنتها بحسابات تعدها يدويا .

(٣٢) أعد حل المشكلة السابقة بالنسبة لبرنامج واحد أو أكثر من البرامج المعطاه فى المشكلة رقم ١ فى الفصل الأول من الكتاب .

(٣٣) يقدم مثالى ٥ – ٣ و ٥ – ٤ صيغتين مختلفتين لبرنامج بسكال يحدد سعر بيع الفطائر عند معرفة حجم الفطائر وعدد المحتويات الإضافية . اختر الصيغة التى تناسب وسط التشغيل المتاح لك . أدخل البرنامج فى الكمبيوتر ، وخزنه ، ثم قم بتنفيذ البرنامج ، مستخدما مجموعات مختلفة من بيانات المدخلات . استخدم النتائج المحسوبة فى إعداد قائمة مثل التى تتوقع أن تراها فى محل الفطائر ، موضحا أسعار كل نوع من أنواع الفطائر بالنسبة للحجم ، وعدد المحتويات الإضافية .

(٣٤) اكتب برنامج بسكال كاملا لكل موقف من المواقف التالية . ادخل كل برنامج فى الكمبيوتر بعد التأكد من صحة أخطاء كتابة البرنامج . خزن البرنامج ، ثم قم بطباعة قائمة به . عندما تتأكد من أن البرنامج قد تم إدخاله بطريقة صحيحة ، قم بترجمة البرنامج وتنفيذه . أعد تكرار ذلك كلما دعت الحاجة لذلك ، حتى تحصل على برنامج خال من الأخطاء . استخدم أسلوب البرمجة التقليدى كلما كان ذلك ممكنا .

أ ) اطبع HELLO فى أول السطر .

ب ) اجعل الكمبيوتر يطبع .

```
HI, WHAT'S YOUR NAME?
```

فى سطر واحد . عند ذلك يقوم المستفيد بإدخال اسمه بعد علامة الاستفهام مباشرة . عند ذلك يترك الكمبيوتر سطرين فارغين ، ويطبع :

```
WELCOME (name)

LET'S BE FRIENDS!
```

على سطرين متتاليين . افرض أن اسم المستفيد يحتوى على 6 خانات بالضبط ( أضف فراغات فى نهاية الاسم إذا كانت هناك حاجة لذلك ) .

ج ) حول قراءات درجة الحرارة من درجات فهرنهيتية إلى درجات مئوية ، مستخدما العلاقة التالية :

$$C = (5/9)*(F-32)$$

اختبر البرنامج ، مستخدما القيم التالية 22- , 200- , 68 , 212 , 150 درجة فهرنهيتية .

د ) حدد كمية النقود الموجودة فى خزينة أحد البنوك تحتوى على $n_1$ من الريال الفضة ، و $n_2$ من نصف ريال ، و $n_3$ من ربع ريال ، و $n_4$ من عشرة هللة ، و $n_5$ من خمسة هللة . استخدم القيم التالية لاختبار البرنامج : $n_3 = 3 , n_2 = 7 , n_1 = 11 , n_5 = 17 , n_4 = 12$ .

هـ ) احسب حجم الكرة ومساحتها ، مستخدما الصيغتين التاليتين :

$$V = \frac{4\pi r^3}{3}$$
$$A = 4\pi r^2$$

اختبر البرنامج ، مستخدما القيم التالية للقطر : 0.2 و 6 , 12.2 .

و ) احسب كتلة الهواء الموجودة في عجلة سيارة ، مستخدما العلاقة التالية :

$$PV = 0.37m(T + 460)$$

حيث P = الضغط بالرطل على البوصة المربعة .

V = الحجم بالقدم المكعب .

m = كتلة الهواء بالرطل .

T = درجة الحرارة بالدرجات الفهرنهيتية .

وتحتوى العجلة على 2 قدم مكعب من الهواء . افرض أن الضغط هو 32 رطلاً على البوصة المربعة في درجة حرارة الغرفة .

ز ) اقرأ كلمة مكونة من خمسة حروف في الكمبيوتر ، ثم قم بعمل شفرة للكلمة حرفا حرفا ، وذلك بطرح 30 من القيمة العددية المستخدمة في تمثيل كل حرف . وعلى هذا ... فإذا كانت شفرة ASCII مستخدمة في تمثيل الرموز ، فإن الحرف a ( والذى يمثل بالعدد 97 ) سوف يصبح C ( والممثل بالعدد 67 ) وهكذا .

اكتب صيغة شفرة الكلمة كمخرجات . اختبر البرنامج مستخدما الكلمات التالية : , Japan , roses , white zebra . ملاحظة : لاحظ أن ( C = chr (ord(a) - 30 .

ح ) اقرأ الكلمة المكونة من خمسة حروف في الكمبيوتر ، والتي سبق عمل تخطيط شفرة لها في الحالة السابقة . قم بعمل شفرة للكلمة بإعادة الخطوات السابقة بصورة عكسية ، ثم قم بطباعة الكلمة الجديدة .

# الفصل السادس

# مكونات التحكم

# Control Structures

فى كل البرامج التى تعرضنا لها حتى الآن كانت تنفذ كل التعليمات مرة واحدة بنفس ترتيب ظهورها فى البرنامج . ومثل هذه البرامج تكون بسيطة وغير واقعية ، حيث إنها لاتحتوى على أى منطق لمكونات التحكم مثل اختبارات تحديد ماإذا كانت إحدى الشروط صحيحة أو خاطئة أو تكرار تنفيذ مجموعة من العبارات أو اختيار إحدى مجموعات العبارات من عدة إمكانيات مختلفة ، إلا أن معظم البرامج العملية تستخدم مثل هذه المعالم استخداما كثيرا . وعلى هذا ... يجب أن نتعلم كيفية استخدام منطق مكونات التحكم فى برامجنا ، حتى يمكن أن نأخذ فى الاعتبار مواقف مشاكل أكثر واقعية وأكثر أهمية .

فمثلا ، تتطلب العديد من البرامج تكرار تنفيذ مجموعة من التعليمات المتتابعة ، حتى يتحقق أحد الشروط المنطقية . وبصفة عامة ... فإن عدد التكرارات المطلوب لايكون معروفا مسبقا . ومثل هذا التكرار يعرف بأنه دورة شرطية conditional looping . كما أنه هناك عملية أخرى هى الدورة غير الشرطية unconditional looping ( أو الدورة looping المسطحة ) والتى يحدث فيها تكرار مجموعة من التعليمات المتتالية عدد محدد من المرات . ويوجد موقف آخر يتكرر ظهوره ، وهو الحاجة إلى إجراء اختبار منطقى ، ثم أخذ بعض الإجراءات الخاصة التى تعتمد على ناتج هذا الاختبار . وهذا مايعرف بالتنفيذ الشرطى conditional execution . وأخيرا هناك نوع خاص من التنفيذ الشرطى ، حيث يتم فيه اختيار مجموعة معينة من العبارات ، وذلك من العديد من المجموعات المتاحة . ويشار فى بعض الأحيان لذلك بالاختيار selection .

ويمكن أداء كل هذه العمليات بسهولة فى البسكال . وسوف نرى كيف يتحقق ذلك فى الفصل الحالى . واستخدام هذه المادة يفتح الباب لمشاكل برمجة أوسع كثيرا وأكثر أهمية .

## ١ – مبادئ : 1. PRELIMINARIES

قبل اعتبار تفاصيل مكونات التحكم فى البسكال ، دعنا نراجع مفاهيم معينة سبق تقديمها فى الفصل الثانى والفصل الثالث . ويجب استخدام هذه المفاهيم مع مكونات التحكم . وعلى هذا ... فإن فهمهم ضرورى قبل الاستمرار فى المزيد .

تذكر أولا أن تعبير البوليان يمثل شرطًا وهذا الشرط إما أن يكون صحيحا أو خاطئا ( انظر قسم ١٠ من الفصل الثانى ) . وتتكون تعبيرات بوليان من مجموعة من العناصر من نفس النوع ( أى نوع باستثناء النوع البوليانى ) مع واحد أو أكثر من المؤثرات العلاقية relational operators السبعة هى : IN , >= , > , <= , < , <> , = ( انظر قسم ٤ من الفصل الثالث ) . وحتى الآن ناقشنا أول ستة مؤثرات علاقية فقط ، أما المؤثر السابع (IN) ، فيختلف عنها ، ولن نتعرض له حتى الفصل الثانى عشر من الكتاب .

مـــثال (٦-١)

فيما يلى العديد من تعبيرات بوليان :

(انظر الصفحة القادمة)

```
count <= 100
sqrt(a+b+c) > 0.005
answer = 0
balance >= cutoff
ch1 < 'T'.
```

تحتوى أول أربعة تعبيرات على عناصر عددية ، وفى الواقع يجب أن تكون قراءتها واضحة .

يفترض فى التعبير الأخير أن ch1 متغير حرفى . ويكون هذا التعبير صحيحا إذا ماأتت الرموز الممثلة بواسطة ch1 قبل T فى فئة الرموز ، إلا إذا ماكانت ord(ch1) < ord('T') ، وإلا فإن التعبير يكون خاطئا .

ويحتوى البسكال بالإضافة إلى المؤثرات العلاقية على مؤثرات منطقية logical operators ( NOT , AND , OR ) تستخدم مع عناصر بوليان ( قسم ٤ من الفصل الثالث ) . أول مؤثرين (AND , OR) يستخدمان مع عناصر بوليان لتكوين تعبيرات منطقية . أما الثالث (NOT) ، فهو بادئة تستخدم فى نفى عنصر بوليان .

**مثـــال (٦-٢)**

فيما يلى تعبيرات بوليان توضح استخدام المؤثرات المنطقية .

```
(count <= 100) AND (ch1 <> '*')
(balance < 1000.0) OR (status = 'R')
(answer < 0) OR ((answer > 5.0) AND (answer < 10.0))
(pay >= 1000.0) AND (NOT single)
```

لاحظ أن status , ch1 متغيران حرفيان فى هذه الأمثلة ، كما أن single هو متغير بوليان . أما بقية المتـغيرات ، فهى عددية ( إما حقيقية أو صحيحة ) .

لاحظ أيضا أن عناصر بوليان موضوعة بين قوسين ، لتجنب أى غموض فى ترتيب التنفيذ .

تذكر أيضا أنه هناك نوعين أساسيين من العبارات فى البسكال ، وهما : النوع البسيط simple والنوع المرتب structured ( انظر قسم ٢ من الفصل الحادى عشر ) تشير العبارات البسيطة إلى عبارات تحديد وأدلة إجراءات أو عبارات GOTO . وسبق أن ناقشنا بالفعل عبارات التحديد ودلائل الإجراءات فى قسم ٢ من الفصل الثالث ، وقسم ٢ من الفصل الثانى عشر . وسوف تناقش عبارات GOTO فى نهاية هذا الفصل .

والعبارات المرتبة مهمة جدا فى هذا الوقت ، وذلك بسبب وقوع مكونات التحكم فى البسكال فى هذه الفئة . وسوف تقدم مناقشة تفصيلية لكل من هذه المكونات فيما بعد فى هذا الفصل . دعنا على أية حال نتذكر الآن المناقشة السابقة لأحد أنواع العبارات المرتبة ، وهى العبارة المركبة .

تحتوى العبارة المركبة compound statement على تتابع من اثنين أو أكثر من العبارات المتتالية والمحصورة بين الكلمتين الأساسيتين END , BEGIN ، مع فصلها عن بعضها بواسطة فواصل منقوطة . والعبارات الفردية التى تتكون منها العبارة المركبة يمكن أن تكون هى نفسها بسيطة أو مرتبة . وعلى هذا ... فإن العبارة المرتبة الواحدة ( وهى مكونات التحكم ) يمكن أن تتواجد داخل عبارة مركبة . والأكثر من ذلك ... يمكن أن ترحل العبارة المركبة داخل عبارة مركبة أخرى .

مـثال (٦-٣)

فيما يلى عبارة مركبة واحدة سبق لنا رؤيتها من قبل :

```
BEGIN
   read(radius);
   area := 3.14159*sqr(radius);
   write(radius,area)
END
```

وفيما يلى عبارة مركبة أكثر تعقيدا :

```
BEGIN
   sum := 0;
   FOR count := 1 TO n DO
      BEGIN
         read(x);
         sum := sum+x
      END;
   write(' sum = ',sum)
END
```

لاحظ أن العبارة المركبة الثانية تحتوى على مكون FOR للتحكم . ( سوف نناقش تفاصيل مكون التحكم هذا فيما بعد فى هذا الفصل – انظر قسم ٤ من هذا الفصل ) . لاحظ أيضا أن العبارة المركبة الأصغر تم ترحيلها داخل مكون FOR للتحكم . وعلى هذا ... فإننا نرى مثالا لعبارة مركبة مرحلة داخل مكون تحكم موجود داخل عبارة مركبة أخرى .

مكونات التحكم الموجودة فى هذا الفصل تستخدم تعبيرات بوليان ، والعبارات المركبة استخداما واسعا .

## ٢ – مكون « بينما ... أعمل » : 2. THE WHILE-DO STRUCTURE

مكون WHILE - DO هو مكون تحكم تكرارى يستخدم لإجراء دورات شرطية . والصيغة العامة لهذا المكـون هـــى :

```
WHILE boolean expression DO statement
```

جزء العبارة من المكون يتكرر تنفيذه ، طالما أن قيمة تعبير بوليان تظل صحيحة . ويمكن أن تكون هذه العبارة بسيطة أو مرتبة ، بالرغم من أنها عادة ماتكون عبارة مركبة تحتوى على بعض المعالم التى يمكنها أن تبدل من قيمة تعبير بوليان .

افرض على سبيل المثال أننا نريد أن نكتب الأرقام من 1 إلى 20 على أن يكون هناك رقم واحد فى كل سطر . يمكن تحقيق ذلك بمكون WHILE - DO التالى :

```
digit := 1;
WHILE digit <= 20 DO
   BEGIN
      writeln(digit);
      digit := digit + 1
   END;
```

حيث افترض أن digit متغير صحيح . وعلى هذا ... فإننا نبدأ بقيمة لهذا المتغير مساوية 1 . ونستمر فى كتابة القيمة الحالية للمتغير digit بزيادة قيمته بمقدار 1 ، ثم تكرار نفس الدورة .

التأثير النهائي لمكون WHILE - DO هذا هو تكرار عملية الكتابة ، وزيادة القيمة في النهاية 20 مرة ، لينتج عن ذلك 20 سطرا متتاليا من المخرجات ، يحتوى كل سطر على قيمة صحيحة تبدأ بالرقم 1 في السطر الأول ، وتنتهى بالعدد 20 في السطر الأخير .

وبالمثل افرض أننا نريد أن نحدد مجموع أول عدد n من الأرقام ، حيث n هو متغير صحيح . يمكن أن يتحقق ذلك بكتابة مايلى :

```
sum := 0;
digit := 1;
WHILE digit <= n DO
   BEGIN
      sum := sum + digit;
      digit := digit + 1
   END;
```

كما أن الكتابة التالية متكافئة مع الكتابة السابقة :

```
sum := 0;
digit := 1;
WHILE digit < n+1 DO
   BEGIN
      sum := sum + digit;
      digit := succ(digit)
   END;
```

حيث digit , sum وكذلك n متغيرات صحيحة .

في أي حالة من الحالتين نبدأ بقيمة تساوى 0 للمتغير sum ، ثم نستمر في إضافة قيم متتالية لـ digit الى sum. وتستمر عملية إضافة القيمة الحالية للمتغير digit إلى sum. وبعد ذلك زيادة قيمة digit بمقدار 1 ، طالما أن ( WHILE ) القيمة الحالية للمتغير digit أقل من n+1 . وفي نهاية هذه العملية يقدم sum مجموع أول n رقما أى :

$$1 + 2 + 3 + \cdots + n.$$

ويستخدم مكون WHILE-DO بكثرة في البسكال ، حيث يوجد هناك العديد من تطبيقات البرمجة التى تتطلب هذا النوع من إمكانية الدورات الشرطية . وسوف نرى العديد من عينات البرامج التى تستخدم مكون WHILE-DO خلال هذا الكتاب .

## مثال (٦-٤)

حساب متوسط قائمة من الأعداد . دعنا نستخدم مكون WHILE-DO في الحصول على متوسط قائمة من الأعداد . وتعتمد هذه الطريقة على استخدام المجموع الجزئي الذى توضع له قيمة ابتدائية مساوية للصفر ، وبعد ذلك يتم تجديده كلما حدثت قراءة لعدد جديد داخل الكمبيوتر . وعلى هذا ... فإن المشكلة تحتاج إلى تكرار .

يمكن أداء الحسابات الفعلية كما يلى :

(١) تحديد قيمة تساوى 1 للمتغير الصحيح Count ( يستخدم هذا المتغير كعداد للدورة )

(٢) تحديد قيمة تساوى 0 للمتغير الحقيقي Sum .

(٣) قراءة قيمة n .

(٤) تكرار أداء الخطوات التالية ، طالما أن قيمة العداد لاتتعدى n ( أى أثناء احتواء المتغير Count على قيمة أقل من n+1 ) .

(أ) قراءة أحد الأعداد الموجودة فى القائمة ( يمثل كل عدد بمتغير حقيقى x ) .

(ب) إضافة العدد إلى قيمة Sum .

(ج) زيادة قيمة Count بمقدار 1 .

(٥) قسمة قيمة Sum على n للحصول على المتوسط المطلوب .

(٦) كتابة القيمة المحسوبة للمتوسط .

وفيما يلى برنامج البسكال الذى يؤدى ذلك .

```
PROGRAM average1(input,output);

(* This program calculates the average of n numbers
   using a WHILE - DO structure. *)

 VAR n,count : integer;
     x,sum,average : real;

BEGIN   (* action statements *)
   count := 1;
   sum := 0;
   readln(n);
   WHILE count < n+1 DO
      BEGIN
         readln(x);
         sum := sum+x;
         count := count+1
      END;   (* count < n+1 *)
   average := sum/n;
   writeln(' The average is ',average)
END.
```

لاحظ أن مكون WHILE-DO يحتوى على عبارة مركبة تتسبب فى ازدياد قيمة Count ، وذلك بالإضافة إلى عبارات أخرى . وفى واقع الأمر ... فإن هذا يتسبب فى جعل تعبير بوليان :

```
count < n+1
```

خاطئا فى إحدى المرات ، وبالتالى يفصل البرنامج عن الكمبيوتر .

لاحظ أيضا أن الدورة لن تنفذ على الإطلاق ، إذا ماحددت قيمة أقل من واحد للمتغيرn . وبالطبع هذا شئ واقعى .

لاحظ أخيرا أن العبارة المركبة الداخلية جدا تم ترحيلها داخل مكون WHILE-DO . ويتسبب ذلك فى جعل قراءة مكون WHILE-DO سهلة .

## ٣ – مكون « كـرر .. حتى » : 3. THE REPAT-UNTIL STRUCTURE

مكون REPEAT - UNTIL هو مكون تحكم للتكرار ، يستخدم لإجراء دورة شرطية . وهو شبيه بمكون WHILE-DO . ويكمل هذان المكونان بعضهما البعض فى بعض المواقف . والصيغة العامة لمكون REPEAT - UNTIL هى :

```
REPEAT sequence of statements UNTIL boolean expression
```

يتكرر تنفيذ مجموعة العبارات حتى يتحقق شرط بوليان . لاحظ أن مجموعة العبارات تنفذ مرة واحدة على الأقل دائما ، حيث إن تعبير بوليان لايتم اختباره إلا عند الانتهاء من مكون التحكم . ( وهذا على عكس مكون WHILE-DO الذى لاينفذ على الإطلاق إذا كانت القيمة الابتدائية لتعبير بوليان خاطئة ) .

لاحظ أن هذا المكون يسمح بوجود مجموعة عبارات ، بينما مكون WHILE-DO لايسمح إلا بعبارة واحـدة فقـط ( ومن الممكن أن تكون عبارة مركبة ) . ومجموعة العبارات الموجودة داخل مكون التحكم REPEAT - UNTIL ليست فى حاجة لأن توضع بين BEGIN , END . وعلى هذا ... فإن الكلمتين الرئيسيتين REPEAT - UNTIL تعملان عمل الأقواس التى تحدد بداية ونهاية تسلسل العبارة . ومن الممكن أن توجد جملة مركبة أو مكون تحكم آخر بالطبع داخل مجموعة الجمل . بالإضافة إلى ذلك ... عادة ماتحتوى مجموعة العبارات بعض معالم يمكنها أن تغير من قيـمة تعـبير بوليان .

ولتوضيح استخدام مكون Repeat - until اعتبر مرة أخرى مشكلة كتابة الأرقام من 1 إلى 20 ، على أن يكتب كل رقم على سطر منفصل . سبق أن رأينا بالفعل كيف يمكن تحقيق ذلك باستخدام مكون WHILE - DO ( انظر قسم ٢ من هذا الفصل ) . والآن دعنا نستخدم مكون REPEAT - UNTIL لنفس الغرض .

```
digit := 1;
REPEAT
   writeln(digit);
   digit := digit + 1
UNTIL digit > 20;
```

مرة أخرى فإن digit يكون متغيرا صحيحا

نبدأ مرة أخرى بقيمة تساوى 1 للمتغير digit ، ثم نستمر فى كتابة القيمة الحالية للمتغير digit ، ونزيد قيمته بمقدار 1 ، ثم تعاد نفس الدورة مرة أخرى . وتستمر العملية حتى UNTIL تتعدى القيمة الحالية للمتغير digit ( والذى يكون قد ازدادت قيمته توا ) القيمة 20 .

والتأثير النهائى لمكون REPEAT - UNTIL هو نفس التأثير المناظر لمكون WHILE - DO سالف الذكر . وعلى هذا ... ينتج 20 سطرا من أسطر المخرجات المتتالية ، وفى كل سطر تظهر قيمة أحد الأرقام المتتابعة .

دعنا نعتبر الآن استخدام مكون REPEAT - UNTIL فى تحديد مجموع أول عدد n من الأرقام ، حيث n هو متغير صحيح معروف ( انظر قسم ٢ من هذا الفصل ) . المكون المطلوب هو :

```
sum := 0;
digit := 1;
REPEAT
   sum := sum + digit;
   digit := succ(digit)
UNTIL digit > n;
```

مرة أخرى يكون digit , sum متغيرين صحيحين

وفى هذه الحالة نبدأ بقيمة تساوى صفر للمتغير sum ، ثم نستمر فى إضافة قيم متتالية للمتغير digit الى المتغير sum . وتستمر عملية إضافة القيمة الحالية للمتغير digit إلى sum ، ثم زيادة قيمة المتغير digit بمقدار 1 حتى UNTIL تتعدى القيمة الحالية للمتغير digit مقـــدار n . عـــند ذلـــك يمثـــل sum مجـــموع أول n من الأرقـــام ، أى يمـــثل n + ... + 3 + 2 + 1 .

ويستخدم مكون REPEAT - UNTIL كما يستخدم مكون WHILE - DO بكثرة فى البسكال . وأحيانا يكون اختيار أحد هذين المكونين بسبب الأفضلية الشخصية فقط . وفى تطبيقات أخرى يتأثر الاختيار بالرغبة فى اختبار تعبير بوليان ، إما فى بداية مكون اختيار التحكم أو فى نهايته .

## مـثال (٦-٥)

حساب متوسط قائمة من الأعداد . دعنا نستخدم الآن مكون REPEAT - UNTIL للحصول على متوسط قائمة بالأعداد . وهذه هى نفس المشكلة المذكورة فى مثال ٦ – ٤ مع استخدام WHILE - DO . وعلى هذا ... فإن الطريقة تشبه طريقة مثال ٦ – ٤ ، مع السماح لمكونات تحكم مختلفة .

**يمكن إجراء الحسابات الفعلية كمايلى :**

**(١) تحديد قيمة تساوى 1 للمتغير الصحيح count ( يستخدم هذا المتغير كعداد للدورة ) .**

**(٢) تحديد قيمة تساوى 0 للمتغير الحقيقى sum .**

**(٣) قراءة قيمة n .**

**(٤) تكرار أداء الخطوات التالية حتى تتعدى قيمة count القيمة n**

**(أ) قراءة قيمة أحد الأعداد الموجودة فى القائمة ( يمثل كل عدد بمتغير حقيقى x ) .**

**(ب) إضافة العدد إلى قيمة sum .**

**(ج) زيادة قيمة count بمقدار 1 .**

**(٥) قسمة قيمة sum على n للحصول على المتوسط المطلوب .**

**(٦) كتابة القيمة المحسوبة للمتوسط .**

**لاحظ التشابه بين هذا التخطيط والتخطيط المقدم فى مثال ٦ – ٤**

**وبرنامج البسكال المناظر لذلك هو مايلى :**

```
PROGRAM average2(input,output);

(* This program calculates the average of n numbers
   using a REPEAT - UNTIL structure. *)

 VAR n,count : integer;
     x,sum,average : real;

BEGIN   (* action statements *)
   count := 1;
   sum := 0;
   readln(n);
   REPEAT
      readln(x);
      sum := sum+x;
      count := count+1
   UNTIL count > n;
   average := sum/n;
   writeln(' The average is ',average)
END.
```

لاحظ أن مجموعة العبارات المحصورة فى مكون UNTIL , REPEAT تحتوى على عبارة تتسبب فى زيادة قيمة count . وهذا يتسبب – فى واقع الأمر – فى أن يصبح تعبير بوليان count > n صحيحا ليفصل البرنامج عن الكمبيوتر .

ويجب أن يكون مفهوما أن هذه الدورة تنفذ دائما مرة واحدة على الأقل ، حيث إن قيمة تعبير بوليان لاتختبر إلا فى نهاية الدورة .

لاحظ أخيرا أن مجموعة العبارات الموجودة داخل مكون REPEAT - UNTIL مرحلة للداخل ، وذلك لتعريف مدى المكون بسهولة .

ويجب أن يقارن هذا المثال بدقة مع مثال ٦ – ٤ . وكل من المكونين ، سواء أكان مكون WHILE - DO أم مكون REPEAT - UNTIL صحيحان بالنسبة لهذه المشكلة . وفى العديد من الحالات – على أية حال – فإن طبيعة المشكلة تقترح أولوية لأحد هذين المكونين على الآخر .

## 4. THE FOR STRUCTURE ٤ – مكون FOR :

يستخدم مكون FOR لأداء دورات غير شرطية فى البسكال ، أى أن هذا المكون يسمح بتكرار بعض الإجراءات عددا محددا من المرات .

ومكون FOR له صيغتان . الصيغة الأكثر شيوعا ، هى :

```
FOR control variable := value 1 TO value 2 DO statement
```

يمكن أن يكون جزء العبارة من المكون بسيطا أو مركبا ، وذلك بالرغم من أن العبارة المركبة يمكنها أن تحتوى مكونات تحكم أخرى . وتنفذ هذه العبارة لكل قيمة من القيم المتتابعة المحددة لمتغير التحكم . وعدد القيم المحددة لمتغير التحكم يحدد على ذلك عدد مرات تنفيذ العبارة .

ويجب أن يكون متغير التحكم من النوع البسيط من أى نوع من أنواع المتغيرات ، باستثناء النوع الحقيقى . وعادة مايكون متغيرا صحيحا ، أو متغيرا يقوم بتعريفه المستفيد . ( وسوف نناقش المتغيرات التى يعرفها المستفيد فى الفصل الثامن من الكتاب ) . وتحدد قيمة ابتدائية value 1 فى البداية لمتغير التحكم . ويأخذ متغير التحكم القيمة المحددة بواسطة value 1 فى البداية . ويأخذ متغير التحكم القيمة التالية تلقائياً فى كل مرة تتكرر العبارة ، وحتى يأخذ القيمة المحددة بانها value 2 . واذا كان متغير التحكم من النوع الصحيح ، فسوف تزداد قيمته تلقائيا بمقدار 1 فى كل مرة يتم فيها تنفيذ العبارة عدد مرات يساوى الرقم الناتج من باقى طرح value 1+1 من value 2 .

ولتوضيح مكون FOR ، دعنا نعتبر مشكلة كتابة أول 20 رقم صحيح موجبة مرة أخرى ، مع كتابة كل رقم على سطر منفصل . ( سبق أن رأينا كيف يمكن أداء ذلك باستخدام مكون WHILE - DO فى قسم ٢ – ، وباستخدام مكون REPEAT - UNTIL فى قسم ٣ – ) . ونحتاج إلى عبارة واحدة فقط FOR - TO لأداء هذا النشاط . وبصفة خاصة يمكننا كتابة مايلى :

```
FOR digit := 1 TO 20 DO writeln(digit);
```

حيث يكون digit متغيرا صحيحا .

يأخذ المتغير digit فى هذا المثال القيم المتتالية 1 , 2 , 3 , ... , 20 ، متسببا فى تنفيذ الدورة 20 مرة . وأثناء كل مرة من هذه المرات تكتب القيمة الحالية للمتغير digit على سطر منفصل ، كما هو مطلوب تماما .

وبالمثل فإن مشكلة تحديد مجموع أول n من الأرقام ، يمكن أن يعبر عنها كما يلى :

```
sum := 0;    sum := sum + digit;
```

وهنا نبدأ بتحديد قيمة 0 للمتغير sum . وفى كل مرة تنفذ فيها الدورة تضاف القيمة الحالية الموجودة فى digit إلى sum . وعلى هذا ... فإن sum يمثل القيمة المطلوبة ، وهى :

$$1 + 2 + 3 + \cdots + n$$

بعد تكرار تنفيذ الدورة عدد n من المرات .

لاحظ السهولة الكبيرة الناتجة من استخدام مكون FOR ، وذلك بالمقارنة بكل من مكونى WHILE - DO و REPEAT - UNTIL بالنسبة لهذه المشاكل ( انظر قسم ٢ - ، وقسم ٣ - ) . وعادة ماتكون هذه هى الحالة عندما يكون عدد مرات تكرار الدورة معروفا مسبقا .

وهناك قواعد محددة يجب اتباعها فى كتابة عبارة FOR . يمكن أن يعبر عن القيم Value 2 , Value 1 بصفة خاصة بثوابت أو متغيرات وتعبيرات . وعلى أية حال ... يجب أن تكون هاتان القيمتان فى نفس النوع مثل نوع متغير التحكم . كما يجب أن تكون قيمة Value 1 أقل من قيمة Value 2 إذا ماكان مطلوبا تكرار العبارة أكثر من مرة واحدة . ( إذا كانت قيمة Value 1 مساوية لقيمة Value 2 ، فتنفذ العبارة مرة واحدة فقط ، أما إذا كانت قيمة Value 1 أكبر من قيمة Value 2 ، فلن تنفذ العبارة على الإطلاق ) .

## مثـال (٦ - ٦)

إن استخدام مكون FOR أقل تعقيدا فى واقع الأمر عما يبدو من أول مرة . ويتضح ذلك من المثالين التاليين :

(*a*)

```
sum := 0;
FOR count := 1 TO n DO BEGIN readln(x); sum := sum+x END;
writeln(' sum=',sum);
```

(*b*)

```
sum := 0;
FOR count := n TO (3*n+1) DO
   BEGIN
      readln(x);
      sum := sum+x
   END;
writeln(' sum=',sum);
```

يحتوى المثال الأول على عبارة مركبة تنفذ عدد n من المرات . ( لاحظ أن n متغير صحيح ، يفترض أن قيمته معروفة ، وأن x متغير حقيقى ) . وخلال كل مرة من مرات تنفيذ الدورة ، يقرأ عدد جديد ( قيمة جديدة للمتغير x ) داخل الكمبيوتر ، ويضاف إلى sum . وبعد ذلك يكتب مجموع كل الأعداد بعد الانتهاء التام من تكرار الدورة .

في المثال الثانى تعطى القيمة الابتدائية والقيمة النهائية لمتغير التحكم عن طريق متغير صحيح وتعبير صحيح على التوالى . لاحظ أن العبارة المركبة انتشرت الآن لتشغل عدة أسطر ، مع عمل الترحيل المناسب . ( هذه هى الصيغة المفضلة ) .

أما الصيغة الثانية لمكون FOR ، فتشبه الصيغة الأولى ، وذلك باستثناء أن الكلمة الأساسية DOWNTO تحل محل الكلمة الأساسية TO . وعلى هذا ... يمكن كتابة مكون على النحو التالى :

FOR *control variable* := *value 1* DOWNTO *value 2* DO *statement*

والإجراء المتخذ بهذه الصيغة لمكون FOR يشبه الاجراء المتخذ بواسطة الصيغة الأولى ، فيما عدا أن متغير التحكم يتم تقويمه للخلف ، بدلا من تقويمه للأمام . وعلى هذا ... فإذا ماكان متغير التحكم من النوع الصحيح ، فسوف يقل تلقائيا بمقدار 1 ، وذلك من القيمة Value 1 إلى القيمة Value 2 أثناء التكرار المتتالى للدورة . وعلى هذا ... يجب أن تكون قيمة Value 1 أكبر من قيمة Value 2 . ( فإذا ماتساوت القيمتان ، تنفذ العبارة مرة واحدة فقط . أما إذا كانت قـيمة Value 1 أقل من قيمة Value 2 ، فلن تنفذ العبارة على الإطلاق ) .

مــثال (٦-٧)

فيما يلى توضيحًا للصيغة الثانية لمكون FOR :

```
FOR i := 0 DOWNTO -12 DO
   BEGIN
      z := 2*i+5;
      writeln(' i=',i,'   z=',z)
   END;
```

سوف يتسبب هذا المثال فى طباعة 13 سطرا . يحتوى كل سطر على القيمة الحالية للمتغير الصحيح i ، تتبعها القيمة المناظرة من العلاقة $z = 2i + 5$ . لاحظ أن القيم المتتالية للمتغير i تتناقص من $i = 0$ فى أول سطر إلى $i = -12$ فى آخر سطر .

ويجب أن يكون مفهوما أن تقويم Value 1 , Value 2 يحدث مرة واحدة فقط قبل تنفيذ الدورة لأول مرة . وعلى هذا ... فيجب ألا يحاول القارئ أن يغير أى من هاتين القيمتين داخل الدورة . كما يجب أن يحتاط القارئ أيضا بأن لايستخدم متغير الدورة بعد فصل مكون FOR ، أى بعد الانتهاء من تنفيذ مكون FOR ، حيث إنه لايكون معرفا بطريقة طبيعية .

مثـــال (٦-٨)

حساب متوسط قائمة بالأعداد . دعنا نعود الى مشكلة حساب متوسط قائمة الأعداد التى سبق أن تعرضنا لها فى مثال ٦ – ٤ ، ومثال ٦ – ٥ . وسوف نستخدم الآن – على أية حال – مكون FOR لأداء أنشطة الدورة .

(١) تحديد قيمة تساوى 0 للمتغير الحقيقى sum .

(٢) قراءة قيمة للمتغير n .

(٣) أداء الخطوات التالية عدد n من المرات ( أى أن متغير التحكم تتراوح قيمته من 1 إلى n ) .

(أ) قراءة العدد التالى فى القائمة . ( يمثل كل عدد بمتغير حقيقى x ) .

(ب) يضاف هذا العدد إلى قيمة sum .

(٤) قسمة قيمة sum على n للحصول على المتوسط المطلوب .

(٥) كتابة القيمة المحسوبة للمتوسط .

لاحظ التشابه الموجود بين هذا التخطيط والتخطيطين السابقين الموجودين فى مثال ٦ – ٤ ، ومثال ٦ – ٥ . وفيما يلى برنامج البسكال كاملا .

```
PROGRAM average3(input,output);

(* This program calculates the average of n numbers
   using a FOR structure. *)

 VAR n,count : integer;
     x,sum,average : real;

BEGIN   (* action statements *)
   sum := 0;
   readln(n);
   FOR count := 1 TO n DO
      BEGIN
         readln(x);
         sum := sum+x
      END;
   average := sum/n;
   writeln(' The average is ',average)
END.
```

لاحظ أن متغير التحكم count حددت له قيمة ابتدائية تساوى 1 . وتزداد هذه القيمة تلقائيا بمقدار 1 فى كل مرة يتم فيها تنفيذ الدورة ، وذلك حتى تصل قيمة count فى النهاية إلى n . وعلى هذا ... فإن الدورة تنفذ عدد n من المرات بالضبط . لاحظ أيضا أن مجموعة العبارات الموجودة داخل مكون FOR مرحلة للداخل . وهذا يسمح بتعريف المكون ومحتوياته بسهولة .

ويجب أن يقارن القارئ هذه الطريقة لحل المشكلة ، مع كيفية حل نفس المشكلة فى كل من مثالى ٦ - ٤ ، و ٦ - ٥ . ويناسب استخدام مكون FOR هذه المشكلة الخاصة عن استخدام أى من مكونى WHILE - DO أو REPEAT - UNTIL ، حيث إن عدد مرات تكرار الدورة معروف مسبقا . وهناك العديد من المشاكل على أية حال لايكون هذا هو حالها ( انظر مثال ٦ -٢٦، ومشكلة ٦ -٤٥ فى نهاية هذا الفصل ) . فى مثل هذه الحالات قد يكون أى من مكونى WHILE - DO أو REPEAT - UNTIL مناسبا أكثر . وعلى هذا ... فلايجب أن يستخلص القارئ تعليقا عاما عن المنفعة النسبية لمكونات التحكم هذه ، فيجب أن يتم تقويم كل تطبيق طبقا لحالته الخاصة .

## ٥ - مكونات التحكم المتداخلة : 5. NESTED CONTROL STRUCTURES

يمكن أن تتداخل مكونات التحكم واحدا داخل الآخر . وليس هناك حاجة لأن يكون كل من مكون التحكم الداخلى والخارجى من نفس النوع ، إلا أنه من الضرورى أن يكون أحد المكونات موجودا بأكمله داخل المكون الآخر . وبكلمات أخرى ... يجب ألا يبدأ مكون تحكم داخل مكون آخر وينتهى بعد الانتهاء من المكون الثانى .

### مـــثال (٦-٩)

إعادة حساب متوسط قائمة الأعداد . افرض أننا نريد حساب المتوسط للعديد من قوائم الأعداد . إذا ماعرفنا مسبقا عدد القوائم التى سوف يجرى لها حساب المتوسط ، فيمكننا عند ذلك استخدام مكون FOR للتحكم فى عدد مرات تنفيذ دورة حساب المتوسط . ويمكن تحقيق حساب المتوسط الفعلى باستخدام أى مكون من مكونات التكرار الثلاثة ، وهى : WHILE - DO أو REPEAT - UNTIL أو FOR كما سبق توضيحها فى مثال ٦ - ٤ ، ومثال ٦ - ٥ ومثال ٦ - ٨ .

دعنا نستخدم على سبيل المثال مكون REPEAT - UNTIL فى حساب المتوسط ، كما فى حالة مثال ٦ - ٥ وعلى هذا ... فإننا نستمر طبقا لمايلى :

(١) قراءة قيمة للمتغير loopmax ، وهى قيمة صحيحة تحدد عدد القوائم التى سوف يحسب المتوسط لها .

(٢) قراءة متكررة لقائمة أعداد ، وتحديد متوسطها ( أى حساب متوسط للقيم المتتالية لمتغير الدورة loop التى تتراوح من 1 إلى loopmax ) . وتتبع نفس الخطوات المعطاه فى مثال ٦ - ٥ لحساب المتوسط .

وفيما يلى برنامج البسكال الكامل لأداء ذلك .

```
PROGRAM average4(input,output);

(* This program uses nested control structures
   to repeatedly calculate the average of n numbers *)

VAR loop,loopmax,n,count : integer;
    x,sum,average : real;

BEGIN   (* action statements *)
   readln(loopmax);
   FOR loop := 1 TO loopmax DO
      BEGIN
         count := 1;
         sum := 0;
         readln(n);
         REPEAT
            read(x);
            sum := sum+x;
            count := count+1
         UNTIL count > n;
         average := sum/n;
         writeln(' list number ',loop,'   The average is ',average)
      END   (* loop *)
END.
```

لاحظ طريقة تداخل مكون REPEAT - UNTIL داخل مكون FOR ( وهذا سهل رؤيته بسبب الترحيل للداخل ) وعلى هذا ... فإن عملية حساب المتوسط تنفذ بالكامل أثناء المرور خلال الدورة الخارجية .

ويتطلب الكثير من التطبيقات استخدام مكون FOR . فى مثل هذه الحالات يجب أن يستخدم كل مكون OR متغير تحكما مختلفا . ( تذكر أيضا أن كل مكون FOR داخلى يجب أن يكون كله محصورا داخل مكون OR الخارجى ) . والمثال التالى يوضح ذلك .

## مثـــال (٦-١٠)

*عوامل الفائدة المركبة* : نفرض أن P هى مجموع رأس المال المستثمر فى n من السنوات ، بمعدل فائدة i فى المئه فائدة مركبه سنوياً. F مقدار رأس المال المتجمع بعد n من السنوات ، ويمكن حساب بواسطة المعادلة المعروفة التالية :

$$F = P(1 + i/100)^n$$

( وتعرف أيضاً بقانون الفائدة المركبة )

– وسيتم تنفيذ الحسابات بالطريقة التالية : –

١– تخصيص القيمة "صفر" بالمتغير الحقيقى sum .

٢– اقرأ قيمة n .

٣– أداء الخطوات التالية n من المرات (بمعنى للقيم المتتالية لعداد متغير التحكم فى المدى من ١ ⟵ n .

أ – إقرأ العدد التالى فى القائمة ( كل عدد يمثل بالمتغير الحقيقى x ).

ب – اجمع هذا العدد على قيمة المتغير sum .

٤– إقسم قيمة sum على n للحصول على المتوسط المطلوب.

٥– أكتب القيمة المحسوبه للمتوسط.

وفيما يلى برنامج البسكال كاملا :

```
PROGRAM compoundinterest(output);
(* This program generates a table of compound interest factors
   to calculate a future value, given a present value or vice versa *)
VAR i,n : integer;
   factor : real;
BEGIN  (* action statements *)
   write(' n     5%       6%       7%       8%');
   writeln('       9%       10%       11%       12%');
   writeln;
   FOR n := 1 TO 20 DO
      BEGIN              (* generate successive rows *)
         write(n:3);
         FOR i := 5 TO 12 DO
            BEGIN              (* generate entries within each row *)
               factor := exp(n*ln(1+0.01*i));
               write(factor:9:5)
            END;
         writeln
      END
END.
```

لاحظ الطريقة التى استخدم بها البرنامج مكون FOR . يتحكم مكون FOR الخارجى فى عدد المضروبات التى تحسب ، بينما يستخدم مكون FOR الداخلى فى حساب كل مضروب من هذه المضروبات . لاحظ أن المكون الداخلى موجود كله داخل المكون الخارجى ، وأن كل من المكونين له متغير التحكم الخاص به . وعلى أية حال ... تستخدم القيمة الحالية لمتغير التحكم الخارجى (n) كقيمة نهائية لمتغير التحكم الداخلى . وتتطلب طبيعة المشكلة أن يستخدم n بهذه الطريقة .

عند تشغيل البرنامج يجب أن يكون هناك حرص بالنسبة لأقصى قيمة تحدد للمتغير n ، حيث إن المضروبات يمكن أن تكون كبيرة جدا ، حتى بالنسبة لقيم متواضعة من قيم n . فإذا ماحددت قيمة كبيرة للمتغير n ، فقد يحدث سريان زائد overflow عند حساب مضروب n . لاحظ بصفة خاصة أن مضروب 8 هو 40.320 . وقد تكون هذه الكمية كبيرة جدا بالنسبة لبعض أجهزة الكمبيوتر الصغيرة .

## 6. THE IF STRUCTURE
## ٦ - مكون إذا :

مكون IF هو مكون تحكم شرطى ، يسمح باتخاذ بعض الإجراءات إذا أخذ أحد الشروط المنطقية قيمة معينة ( صحيح أو خاطئ ) .

وهذا المكون له صيغتان مختلفتان . وأبسط صيغة هى :

IF *boolean expression* THEN *statement*

وعادة مايشار إلى هذا المكون بأنه مكون IF - THEN . وينفذ جزء العبارة من المكون إذا ماكان تعبير بوليان صحيحا فقط . أما إذا كان تعبير بوليان خاطئًا ، فإن جزء العبارة من المكون لاينفذ . ويمكن للعبارة نفسها أن تكون بسيطة أو مرتبة ، بالرغم من أنها عادة ماتكون مركبة .

### مثال (٦-١١)

فيمايلى بعض أمثلة لمكون IF - THEN

```
IF count <= 100 THEN count := count+1;
IF tag = '*' THEN
   BEGIN writeln(accountno); credit := 0 END;
IF test THEN BEGIN x := 100; test := false END;
IF (balance < 1000.0) OR (status = 'R') THEN writeln(balance);
```

المتغير الصحيح count فى المكون الأول يزداد بمقدار 1 إذا لم تتعدَّ القيمة الحالية 100 . ويتسبب المثال الثانى فى طباعة قيمة المتغير accountno ، وتحديد القيمة 0 للمتغير credit إذا ما حددت نجمة "*" للمتغير الحرفى tag . لاحظ أن هذا المثال يحتوى على عبارة مركبة .

ويحتوى المثال الثالث على متغير بوليان ( test ) وعبارة مركبة . فإذا ماكانت القيمة الابتدائية للمتغير test صحيحة ؛ فتحدد القيمة 100 للمتغير x ، ويصبح test خاطئا .

وفى آخر مثال تطبع قيمة balance إذا ماكانت أقل من 1000.0 ، أو إذا ماكان المتغير الحرفى status يمثل الحرف R ( أو إذا ماتحقق الشرطان ) .

وأخيرا ... لاحظ أنه لاتوجد فواصل منقوطة داخل مكون IF - THEN ، فيما عدا الفواصل الموجودة داخل العبارة المركبة .

والصيغة الثانية لمكون IF هى :

IF *boolean expression* THEN *statement 1* ELSE *statement 2*

وعادة مايشار إليها بأنها مكون IF - THEN - ELSE . وفى هذه الحالة تنفذ statement 1 إذا ماكان تعبير بوليان صحيحا ، وإلا فتنفذ statement 2 . لاحظ تنفيذ إحدى العبارتين دائما ( ومن المستحيل تنفيذهما معا ) . مرة أخرى ... العبارة الفردية يمكن أن تكون بسيطة أو مركبة . وعادة ماتكون العباراتان مركبتين .

يجب ألا تظهر الفواصل المنقوطة داخل مكون IF - THEN - ELSE ، إلا إذا استخدمت كفواصل داخل عبارة مركبة . وأحيانا يخطئ المبتدئون بوضع فواصل منقوطة قبل الكلمة الأساسية ELSE . ويجب تجنب ذلك ، حيث إن هذا يتسبب فى خطأ فى الترجمة .

## مثال (٦-١٢)

فيما يلى عدة أمثلة توضح استخدام مكون IF - THEN - ELSE .

```
IF status='S' THEN tax := 0.20*pay ELSE tax := 0.14*pay;

IF tag = '*' THEN BEGIN writeln(accountno); credit := 0 END
          ELSE credit := 1000;

IF circle THEN BEGIN
              readln(radius);
              area := 3.141593*sqr(radius);
              writeln(' Area of circle=',area)
            END
        ELSE BEGIN
              readln(length, width);
              area := length*width;
              writeln(' Area of rectangle=',area)
            END;
```

تحدد قيمة المتغير tax فى المثال الأول بإحدى طريقتين ، طبقا للقيمة المحددة للمتغير الحرفى status .

ينظر المثال الثانى إلى الحسابات التى لها اشارات tagged . فإذا كان الحساب له اشارة ( أى إذا ماحددت نجمة "*" للمتغير الحرفى tag ) ، فيطبع رقم الحساب ويوضع حد المديونية مساويا للصفر ، وإلا فيتحدد حد المديونية بأنه 1000 .

ويبين المثال الثالث كيفية حساب مساحة لأى من شكلين هندسيين ، فإذا كان متغير بوليان circle صحيحا ، تقرأ قيمة نصف القطر داخل الكمبيوتر ، ثم تحسب المساحة وتطبع بعد ذلك . أما إذا كان متغير بوليان circle خاطئا ، يقرأ طول وعرض المستطيل داخل الكمبيوتر ، وتحسب المساحة ، ثم تطبع قيمتها .

لاحظ مرة أخرى أنه لاتوجد فواصل منقوطة فى مكون IF - THEN - ELSE ، فيما عدا الفواصل الموجودة داخل العبارات المركبة .

مكونات IF يمكنها أن تتداخل مع بعضها ، وذلك مثل أى مكونات تحكم أخرى . وفيما يلى بعض الصيغ التى يمكن أن تأخذها مكونات IF فى تداخلها مع بعضها .

الصيغة الأكثر شيوعا هى تداخل الطبقتين ، والتى تأخذ الشكل التالى :

```
IF be1 THEN IF be2 THEN s1 ELSE s2
       ELSE IF be3 THEN s3 ELSE s4
```

حيث bc 3 , bc 2 , bc 1 تمثل تعبيرات بوليان s 4, s 3 , s 2 , s 1 تمثل عبارات . فى هذه الحالة ينفذ مكون IF - THEN - ELSE كاملا إذا ماكانت bc1 صحيحة . وينفذ مكون آخر إذا ماكانت bc1 خاطئة . ومن الممكن بالطبع أن تحتوى s 4 , s 3 , s 2 , s 1 على مكونات IF - THEN - ELSE أيضا . وعلى هذا ... فيمكن أن يكون لدينا تداخل متعدد .

وفيما يلى صيغة أخرى لتداخل الطبقتين :

```
IF be1 THEN s1
      ELSE IF be2 THEN s2 ELSE s3

IF be1 THEN IF be2 THEN s1 ELSE s2
      ELSE s3

IF be1 THEN IF be2 THEN s1 ELSE s2
```

فى كل من الحالة الأولى والثانية تتحدد طبيعة التبعية لمكون IF - THEN - ELSE بواسطة السطر المكتوبة عليه . وفى الحالة الأخيرة ليس من الواضح أى تعبير بوليان هو المصاحب لجزء ELSE . النتيجة هى bc 2 . وعلى هذا ... فإن المثال الأخير يكافئ مايلى :

```
IF be1 THEN
      BEGIN IF be2 THEN s1 ELSE s2 END
```

إذا ماأردنا أن يصاحب جزء ELSE تعبير بوليان bc1 ، بدلا من تعبير بوليان bc 2 ، فيجب أن نكتب مايلى :

```
IF be1 THEN BEGIN IF be2 THEN s1 END
      ELSE s2
```

وعلى هذا ... فيجب إجراء هذا النوع من التداخل بدقة ، وذلك لتجنب أى إبهام ممكن

## مثال (٦-١٣)

إعداد شفرة للرموز . دعنا نكتب برنامج بسكال بسيط يقرأ رموز بشفرة ASCII ، ويخرج شفرة الرموز فى مواقعها . فإذا كان الرمز حرفا أو رقما ، فإنه يحل محله بالرمز التالى له ، فيما عدا الحرف Z الذى يجب أن يحل محله A ، والحرف z الذى يحل محله a ، والرقم 9 الذى يحل محله 0 . وعلى هذا ... فإن 1 تصبح C , 2 تصبح p ,D تصبح q وهكذا . وأى رمز غير الحروف والأرقام تحل محله نجمة (*) . واستمر فى الحسابات حتى يتم إدخال النجمة كرمز مدخلات . ( يجب أن تفسر النجمة على أنها تحقق لشرط التوقف ) .

وحتى يعد تخطيطا للبرنامج ، دعنا نعرف المتغيرات التالية :

charin = متغير حرفى يمثل رمز المدخلات .

charout = متغير حرفى يمثل رمز المخرجات .

flag = متغير بوليان يحدد ماإذا كانت إجراءات الدورة تستمر أم لا .

وتستمر الحسابات كمايلى :

(١) تحديد قيمة صحيح للمتغير flag .

(٢) تكرار الخطوات التالية ، حتى تصبح قيمة flag خاطئًا .

(أ) قراءة قيمة للمتغير charin .

(ب) إذا كانت قيمة charin هى النجمة (*) ، توضع قيمة خطأ للمتغير flag . وإلا فيتحدد إذا ماكان charin يمثل حرفا أم رقما .

١) إذا كان charin يمثل حرفا أو رقما ؛ ضع القيمة المناسبة للمتغير charout .

٢) إذا كان charin يمثل رمزا خاصا ، ولايمثل حرفا أو رقما ؛ ضع نجمة (*) للمتغير charout .

٣) أخرج القيمة الجديدة المحددة للمتغير charout .

وفيما يلى برنامج البسكال المناظر لذلك .

```
PROGRAM charactercode(input,output);

(* THIS PROGRAM REPLACES INPUT CHARACTERS
   WITH EQUIVALENT ENCODED CHARACTERS *)

VAR charin,charout : char;
    flag : boolean;
BEGIN
   flag := true;
   REPEAT
      write(' Enter character: ');
      readln(charin);
      IF charin = '*'   (* stopping criterion *)
         THEN flag := false
         ELSE BEGIN     (* character replacement *)
                 IF ((charin >= '0') AND (charin < '9')) OR
                    ((charin >= 'A') AND (charin < 'Z')) OR
                    ((charin >= 'a') AND (charin < 'z'))
                    THEN charout := succ(charin)
                    ELSE IF charin = '9'
                            THEN charout := '0'
                            ELSE IF charin = 'Z'
                                    THEN charout := 'A'
                                    ELSE IF charin = 'z'
                                            THEN charout := 'a'
                                            ELSE charout := '*';
                 writeln('   New character: ',charout);
                 writeln
              END
   UNTIL flag = false
END.
```

لاحظ أن البرنامج يحتوى على العديد من عبارات IF - THEN - ELSE بمستويات تداخل متعددة . لاحظ أيضا أننا اخترنا استخدام مكون REPEAT - UNTIL لأداء عمليات الدورة . وقد كان من الممكن اختيار مكون WHILE - DO ببساطة ، إلا أن مكون FOR ليس مناسبا فى هذه الحالة .

وينتج عن تنفيذ هذا البرنامج المخرجات التالية . ( موضوع خط تحت استجابات المستفيد ) .

```
Enter character: f
  New character: g

Enter character: z
  New character: a

Enter character: P
  New character: Q

Enter character: 5
  New character: 6

Enter character: 9
  New character: 0

Enter character: ?
  New character: *

Enter character: *
```

## مـثال (٦-١٤)

حل معادلة جبرية . بالنسبة للمحبين للرياضيات ، يوضح هذا المثال كيف يمكن استخدام الكمبيوتر فى حل معادلات جبرية لايمكن حلها بالطرق البدائية . أعتبر على سبيل المثال المعادلة التالية :

$$x^5 + 3x^2 - 10 = 0.$$

لايمكن إعادة ترتيب هذه المعادلة للحصول على حل دقيق للمتغير x . وعلى أية حال ... يمكننا أن نحدد الحل بتكرار عملية المحاولة والخطأ ( أى عملية تكرارية ) والتى تحسن بصفة دائمة من التقدير الأولى للحل .

ونبدأ بإعادة ترتيب المعادلة لتأخذ الشكل التالى :

$$x = (10 - 3x^2)^{1/5}$$

والعملية هى أننا نقدر قيمة المتغير x ، ونعوض بهذه القيمة فى الطرف الأيمن للمعادلة التى أعيد ترتيبها ، وذلك لحساب قيمة x الجديدة ، فإذا ماكانت قيمة x الجديدة مساوية لقيمة x القديمة (.أو قريبة جدا منها ) ، نكون قد حصلنا على حل المعادلة ، وإلا فإننا نعوض بقيمة x الجديدة فى الطرف الأيمن للمعادلة ، لنحصل على قيمة أخرى للمتغير x وهكذا . وتستمر هذه العملية حتى يصبح هناك قيمتان متتاليتان للمتغير x قريبتان جدا من بعضهما ( أى حتى تبدأ الحسابات فى التقارب ) ، أو حتى يتم تنفيذ عدد محدد من التكرارات ( حتى لايستمر تكرار الحسابات عددا طويلا من المرات إذا لم يحدث تقارب للنتائج المحسوبة ) .

ولرؤية كيفية عمل هذه الطريقة ، افرض أننا اخترنا قيمة ابتدائية 10 للمتغير x . بالتعويض بهذه القيمة فى الطرف الأيمن للمعادلة ؛ نحصل على :

$$x = [10 - 3(1.0)^2]^{1/5} = 1.47577$$

ثم نعوض بعد ذلك بقيمة x الجديدة فى الطرف الأيمن للمعادلة لنحصل على :

$$x = [10 - 3(1.47577)^2]^{1/5} = 1.28225$$

وبالاستمرار فى هذه العملية ، فإننا نحصل على :

$$x = [10 - 3(1.28225)^2]^{1/5} = 1.38344$$
$$x = [10 - 3(1.38344)^2]^{1/5} = 1.33613$$

وهكذا . لاحظ أن القيم المتتالية للمتغير x بدأت تتقارب ناحية نتيجة نهائية .

ويعتمد نجاح الطريقة على قيمة التقدير الابتدائى . فإذا ما كانت هذه القيمة كبيرة جدا ، فإن الكمية الموجودة بين القوسين تكون سالبة ، ولايمكن رفع الرقم السالب إلى أس كسرى . وعلى هذا ... فيجب أن نختبر إذا ماكانت القيمة $10 - 3x^2$ سالبة أم لا عند التعويض بقيمة جديدة للمتغير x فى الطرف الأيمن للمعادلة .

ولكى نكتب تخطيطا للبرنامج ، دعنا نعرف الرموز التالية :

count = عداد عدد مرات التكرار ( يتزايد count بمقدار وحدة واحدة فى كل مرة يحدث فيها التكرار ) .

guess = قيمة x التى يعوض بها فى الطرف الأيمن للمعادلة .

root= قيمة x المحسوبة الجديدة .

test = الكمية $(10-3x^2)^{1/5}$ .

error = القيمة المطلقة للفرق بين root و guess .

flag = متغير بوليان يحدد ماإذا كان التكرار يستمر أم لا .

سوف نستمر فى الحسابات ، حتى يتحقق أحد الشروط التالية :

(١) تصبح قيمة error أقل من 0.00001 ( وفى هذه الحالة فإننا نحصل على التقارب ) .

(٢) إتمام 50 تكرارا ( أى يصبح count = 50 ) .

(٣) تكون قيمة المتغير سالبة ( وفى هذه الحالة لايمكن للحسابات أن تستمر ) .

دعنا نتقدم أكثر فى الحسابات عن طريق كتابة كل قيمة للجذر كمخرجات .

والآن يمكننا أن نكتب التخطيط التالى للبرنامج :

(١) وضع قيمة ابتدائية للمتغيرين count , flag ( وضع قيمة صحيح للمتغير flag ، وقيمة 0 للمتغير count ) .

(٢) قراءة قيمة ابتدائية كتقدير للحل ( المتغير guess ) .

(٣) إجراء العمليات التكرارية التالية ، طالما أن قيمة flag هى صحيح :

(أ) زيادة خطوة العداد ( زيادته بمقدار 1 ) .

(ب) تحديد قيمة خطأ للمتغير flag إذا ماكانت القيمة الجديدة للمتغير count مساوية 50 . ( وهذا يحدد آخر مرة لتكرار الدورة ) .

(ج) تقويم test . فإذا ماكانت قيمته موجبة ، يستمر العمل على النحو التالى :

١) حساب قيمة جديدة للمتغير root ، ثم كتابة قيمة count الحالية كمخرجات ، تليها قيمة root الحالية .

٢) تقويم error ( القيمة المطلقة للفرق بين guess , root ) . فإذا ماكانت هذه القيمة أكبر من 0.00001 ، تحدد قيمة root للمتغير guess ، ويستمر العمل بتكرار الدورة ، وإلا فتكتب القيم الحالية لكل من count , root كمخرجات وتوضع قيمة flag خطأ ( وتعتبر القيمة الحالية للمتغير root أنها هى الحل المطلوب ) .

(د) أما إذا كانت قيمة test غير موجبة ، فلايمكن عند ذلك استمرار الحسابات . وعند ذلك تكتب رسالة خطأ مناسبة ( "numbers out of range" ) ، وتوضع قيمة flag خاطئة .

(٤) عند إتمام عمليات الدورة ، تكتب رسالة خطأ مناسبة ( "Convergence not obtained" ) إذا ماكانت قيمة count 50 ، وكانت قيمة error أكبر من 0.00001 .

ودعنا الآن نعبر عن تخطيط البرنامج فى صورة شفرة شبيهة ، وذلك لتبسيط النقل من التخطيط العام إلى برنامج بسكال .

```
PROGRAM equation(input,output);

(* variable declarations *)

BEGIN

(* initialize flag and count *)

(* read guess *)

WHILE flag DO
   BEGIN

      (* increment count *)

      IF count = 50 THEN flag := false;

      (* evaluate test *)

      IF test > 0 THEN BEGIN (* another iteration *)

                         (* evaluate root *)

                         (* write count and root *)

                         (* evaluate error *)

                         IF error > 0.00001 THEN guess := root;
                                           ELSE BEGIN
                                                   flag := false;

                                                   (* write final answer -
                                                   root and count *)

                                                END
                     END

                     ELSE BEGIN
                             flag := false;
                             (* write error message *)
                          END
      END;
     IF (count = 50) AND (error > 0.00001) THEN
                                   (* write error message *)
END.
```

وفيما يلى برنامج البسكال كاملا :

```
PROGRAM equation(input,output);

(* THIS PROGRAM DETERMINES THE ROOTS OF AN ALGEBRAIC EQUATION
   USING AN ITERATIVE PROCEDURE *)
```

(تكملة البرنامج فى الصفحة التالية)

```
VAR count : integer;
    guess,root,test,error : real;
    flag : boolean;
BEGIN
   flag := true;   (* Initialize *)
   count := 0;
   write(' Initial guess= ');   (* Begin input routine *)
   readln(guess);
   writeln;
   WHILE flag DO   (* Begin main loop *)
      BEGIN
         count := count+1;
         IF count = 50 THEN flag := false;
         test := 10-3*sqr(guess);
         IF test > 0
            THEN BEGIN   (* Another iteration *)
                    root := exp(0.2*ln(test));
                    write(' Iteration number ',count:2);
                    writeln('   x= ',root:7:5);
                    error := abs(root-guess);
                    IF error > 0.00001
                       THEN guess := root
                       ELSE BEGIN   (* Write final answer *)
                               flag := false;
                               writeln;
                               write(' Root= ',root:7:5);
                               writeln('   No. of iterations= ',count:2)
                            END
                 END
            ELSE BEGIN   (* Error message *)
                    flag := false;
                    writeln;
                    write(' Numbers out of range - ');
                    writeln('try again using another initial guess')
                 END
      END;
   IF (count = 50) AND (error > 0.00001) (* Error message *)
      THEN BEGIN
              writeln;
              writeln(' Convergence not obtained after 50 iterations')
           END
END.
```

يحتوى البرنامج على عدد من مكونات التحكم . ويتم التحكم بصفة خاصة فى التكرار الفعلى بواسطة مكون WHILE - DO . وكان من الممكن استخدام مكون REPEAT - UNTIL بدلا منه أيضا . وبالإضافة إلى ذلك ... يوجد العديد من مكونات IF - THEN , IF - THEN - ELSE داخل البرنامج . لاحظ استخدام مكونات IF - THEN - ELSE المتداخلة ناحية منتصف البرنامج .

لاحظ العبارة التالية :

```
root := exp(0.2*ln(test));
```

والموجودة بقرب منتصف البرنامج . هذه العبارة مطلوبة لترفع قيمة test إلى القوة 0.2 ( تذكر أن البسكال لايحتوى على عملية أسية ) . وعلى هذا ... فإننا نحصل على اللوغاريتم الطبيعي log للمتغير test ، ونضربه في 0.2 ، ثم نرفع e للقوة المساوية لنتيجة حاصل الضرب .

والمخرجات الناتجة من اختيار قيمة تقديرية أولية تساوى 1 للمتغير x موضحة أدناه . ( وإجابات المستفيد موضوع تحتها خط ) . لاحظ أن الحسابات التى حدث لها تقارب إلى الحل بأن x مساوية 1.35196 ، وذلك بعد 16 تكرارا ، وتوضح المخرجات المطبوعة أن القيم المتتالية للمتغير x أصبحت متقاربة أكثر ؛ مما أدى إلى الحل النهائى .

```
Initial guess= 1

Iteration number  1   x= 1.47577
Iteration number  2   x= 1.28225
Iteration number  3   x= 1.38344
Iteration number  4   x= 1.33613
Iteration number  5   x= 1.35952
Iteration number  6   x= 1.34826
Iteration number  7   x= 1.35375
Iteration number  8   x= 1.35109
Iteration number  9   x= 1.35238
Iteration number 10   x= 1.35175
Iteration number 11   x= 1.35206
Iteration number 12   x= 1.35191
Iteration number 13   x= 1.35198
Iteration number 14   x= 1.35195
Iteration number 15   x= 1.35196
Iteration number 16   x= 1.35196

Root= 1.35196 . No. of iterations= 16
```

افرض الآن أن القيمة 10 تم اختيارها للمتغير x كتقدير ابتدائى . ينتج عن هذا الاختيار قيمة سالبة للمتغير test فى أول محاولة . وعلى هذا ... تظهر المخرجات على النحو التالى :

```
Initial guess= 10

Numbers out of range - try again using another initial guess
```

ومن الممتع رؤية ماذا يحدث عندما يكون التقدير الابتدائى مساويا 1 ، مع تقليل أقصى عدد لمرات التكرار من 50 إلى 10 مرات . ونشجع القارئ بأن يحاول عمل ذلك ، مع ملاحظة النتائج .

## ٧ – مكون الحالة : 7. THE CASE STRUCTURE

مكون *الحالة* هو مكون تحكم شرطى يسمح باختيار مجموعة عبارات محددة من عدة مجموعات متاحة . ويعتمد الاختيار على القيمة الحالية لتعبير يشار إليه بأنه القائم بالاختيار selector .

والصيغة العامة لمكون *الحالة* هى :

```
CASE expression OF
     case label list 1 : statement 1;
     case label list 2 : statement 2;
          .
          .
          .
     case label list n : statement n
END
```

يمكن أن يكون التعبير بسيطا ، على ألا يكون حقيقيا . وعادة مايأخذ صورة متغير فردى بسيط .

وكل عنوان label من عناوين الحالة يمثل إحدى القيم المسموح بها للتعبير . وعلى هذا ... فإذا كان التعبير من النوع الصحيح ، فإن عناوين labels الحالة تمثل قيما صحيحة تقع داخل مدى مسموح به . ولاتحتاج عناوين الحالة لأن تكون مرتبة ترتيبا معينا ، بالرغم من أن كل عنوان من هذه العناوين يجب أن يكون فرديا . والأكثر من هذا ... يمكن أن يظهر كل عنوان فى قائمة list واحدة فقط . ( ملاحظة : يشار إلى هذه العناوين بأنها عناوين الحالة case labels ، وذلك لتمييزها عن أنواع العناوين الأخرى ، والتى سوف تناقش فى القسم التالى ) .

ويمكن أن تكون العبارات بسيطة أو مرتبة . ومن الشائع جدا استخدام العبارات المركبة . ويسمح أيضا باستخدام العبارة الخالية لتحديد أنه لن تتخذ أى إجراءات لقيم معينة من قيم القائم بالاختيار selector . ولاتحتاج العبارات لأن تكون فردية ( أى أنه يمكن استخدام نفس العبارة مع قائمتين أو أكثر من عناوين الحالة ) .

وتنفذ العبارة إذا ماتوافقت عناوين الحالة المناظرة لها فقط مع القيمة الحالية للتعبير . وعلى هذا ... فإن القيمة الحالية للتعبير تحدد العبارة التى سوف تنفذ . فإذا لم تتوافق القيمة الحالة للتعبير مع أى عناوين ، فلايكون الإجراء معرفا . ( بعض صيغ البسكال تحتوى على جزء " وإلا otherwise " والذى يحدد الإجراء الذى يتخذ إذا لم تتوافق قيمة القائم بالاختيار selector مع أى من العنوانين ) .

## مـثال (٦-١٥)

فيما يلى مكون *حالة* تقليدى . وفى هذه الحالة ... فإن choice من النوع الحرفى :

```
CASE choice OF
     'R' : writeln(' RED ');
     'W' : writeln(' WHITE ');
     'B' : writeln(' BLUE ')
END;
```

يطبع RED إذا ماتحددت R كقيمة للمتغير choice . وتظهر WHITE إذا ماحددت W كقيمة للمتغير choice ، كما تكتب BLUE إذا ما تحددت B كقيمة للمتغير choice . ولاينتج أى مخرجات إذا ماتحدد أى رمز آخر للمتغير choice غير الحروف R أو W أو B .

## مـثال (٦-١٦)

فيما يلى مثالاً تقليديًا آخر لمكون *الحالة* :

```
CASE trunc(x/10) OF
     1   : y := y+5;
     3,5 : y := y-2;
     6   : y := 2*(y+1);
     2,4 : ;
     9   : y := 0
END;
```

فى هذا المثال y , x من النوع الحقيقى ، ويحذف الكسر العشرى من قيمة x / 10 . ويستخدم الجزء الصحيح كقائم بالاختيار . فإذا كان الجزء الصحيح مساويا 1 ، فسوف تزداد قيمة y بمقدار 5 . وبالمثل فإن قيمة y تقل بمقدار 2 إذا ماكان الجزء الصحيح مساويا 3 أو 5 . أما إذا كان الجزء الصحيح مساويا 6 ، فتتحدد قيمة y بأنها (y+1) *2

كما تحدد قيمة y بأنها 0 إذا ماكان الجزء الصحيح مساويا 9 . وتظل قيمة y كما هى بدون تغيير إذا ماكانت قيمة الجزء الصحيح 2 أو 4 . ولاتكون الإجراءات معرفة لكل قيم القائم بالاختيار الأخرى ، أى أنه لاتتخذ أية إجراءات أخرى ( ويمكن إنتاج رسالة خطأ فى هذه الحالة مثلا ) .

ومن الناحية العملية ... يمكن التفكير فى مكون الحالة كبديل لاستخدام مكونات IF - THEN - ELSE . وعلى أية حال ... فإن هذا المكون يمكنه أن يحل محل مكونات IF - THEN - ELSE التى تختبر المعادلات فقط . وفى هذه الحالة عادة مايكون استخدام مكون الحالة مريحا .

## مثـــال (٦-١٧)

حساب الاستهلاك . دعنا نأخذ فى الاعتبار كيف يحسب الاستهلاك السنوى لبعض العناصر المستهلكة ( مثل المبنى أو المكنة وغيرهما ) . وهناك ثلاث طرق شائعة الاستخدام لحساب الاستهلاك ، وهى طريقة الخط المستقيم straight-line ، وطريقة القسط المتناقص double declining balance وطريقة مجموع الأرقام الدالة على السنوات sum-of-the-year's digits . ونريد كتابة برنامج بسكال يسمح لنا باختيار أى طريقة من هذه الطرق لكل مجموعة من الحسابات .

وتبدأ الحسابات بقراءة القيمة الأصلية ( التى لم يحدث لها استهلاك ، وتعرف بأنها القيمة الابتدائية أيضا ) للعنصر والعمر الافتراضى للعنصر ( أى عدد السنوات المتوقع أن يستهلك خلالها العنصر ) ورقم يحدد أى الطرق الثلاث لحساب الاستهلاك هى المستخدمة . وبعد ذلك يحسب الاستهلاك السنوى والقيمة المتبقية ( غير المستهلكة ) للعنصر وتكتب ، وذلك لكل سنة .

طريقة الخط المستقيم هى أسهل الطرق الثلاث . وفى هذه الطريقة تقسم القيمة الأصلية للعنصر على عمر العنصر الافتراضى ( إجمالى عدد السنين ) . وخارج القسمة هو الكمية المستهلكة فى كل سنة ، فإذا ما كان أحد العناصر له قيمة أصلية 8000 دولار وعمره الافتراضى عشر سنوات ، فإن قيمة الاستهلاك السنوى تكون 800 دولار ، بحيث أن العنصر تقل قيمته بمقدار 800 دولار سنويا . لاحظ أن الاستهلاك السنوى ثابت فى كل السنوات .

وعند استخدام طريقة القسط المتناقص ، فإن قيمة العنصر تقل بنسبة سنوية ثابتة . ( وعلى هذا ... فإن القيمة الفعلية للاستهلاك بالدولار تتغير من سنة لأخرى ) ، وللحصول على معامل الاستهلاك ، فإننا نقسم 2 على العمر الافتراضى للعنصر ، ثم يضرب هذا المعامل فى قيمة العنصر فى بداية كل سنة ( وليس فى القيمة الأصلية أو القيمة الابتدائية للعنصر ) ، وذلك للحصول على قيمة الاستهلاك السنوى .

افرض على سبيل المثال أننا نريد حساب استهلاك عنصر قيمته الأصلية 8000 دولار على مدار 10 سنوات باستخدام طريقة القسط المتناقص . معامل الاستهلاك يكون 2/10 = 0.20 ، وعلى ذلك ... فإن قيمة الاستهلاك فى السنة الأولى هى 0.2 x 8000 = 1600 دولار . وقيمة الاستهلاك فى السنة الثانية هى = (8000- 1600) 0.2 = 0.2 x 6400 = 1280 دولار . وقيمة الاستهلاك فى السنة الثالثة هى 0.2 x 5120 = 1024 دولارًا ، وهكذا .

وفى طريقة مجموع الأرقام الدالة على السنوات تقل قيمة العنصر بنسبة تختلف من سنة لأخرى . ويكون معامل الاستهلاك كسرا مقامه عبارة عن مجموع الأرقام من 1 إلى n ، حيث n هى العمر الافتراضى للعنصر ( فإذا كان العمر الافتراضى 10 ، فإن المقام يكون (1+2+3+...+10 = 55) . وبالنسبة للسنة الأولى يكون البسط n . وبالنسبة للسنة الثانية يصبح البسط n-1 . وفى السنة الثالثة n-2 وهكذا . ويتم الحصول على قيمة الاستهلاك السنوى بضرب معامل الاستهلاك فى القيمة الأصلية للعنصر .

ولمعرفة كيف تعمل طريقة جمع الأرقام الدالة على السنين ، فإننا نحسب استهلاك نفس العنصر الذى قيمته الابتدائية 8000 دولار ، وعمره الافتراضى 10 سنوات . قيمة الاستهلاك للسنة الأولى هى 8000 x (10/55) = 1454.55 دولار ، وللسنة الثانية 8000 x (9/55) = 1309.09 وهكذا .

دعنا الآن نعرف الرموز التالية :

val = قيمة العنصر

tag = القيمة الأصلية للعنصر ( أى القيمة الأصلية للمتغير val )

deprec = قيمة الاستهلاك السنوى

n = العمر الإفتراضى للعنصر

year = عداد تتراوح قيمتة من 1 إلى n

choice = رقم صحيح يحدد الطريقة المستخدمة لحساب الاستهلاك

flag = متغير بوليان يحدد ماإذا كانت مجموعة الحسبات تنفذ أم لا

ويتبع برنامج البسكال التخطيط التالى :

(١) توضيح المتغيرات .

(٢) تحديد قيمة صحيح للمتغير flag .

(٣) تكرار الخطوات التالية ، حتى تصبح قيمة flag خطأ :

(أ) قراءة قيمة للمتغير choice ( أما 1 أو 2 أو 3 أو 4 لتحدد نوع الحسابات التى تستخدم ) .

(ب) إذا لم تكن قيمة choice هى 4 ، تقرأ قيم n , val ( إذا كانت قيمة choice هى 4 ، فإن هذا يحدد نهاية التنفيذ ) .

(ج) إذا كانت قيمة choice هى 1 أو 2 أو 3 ، يحسب الاستهلاك السنوى والقيمة الجديدة للعنصر طبقا للطريقة المناسبة ، والتى تحدد طبقا لقيمة choice ، وتطبع النتائج سنة بسنة . أولا ( إذا كانت قيمة هى 4 ) ؛ توضع قيمة خطأ للمتغير flag ، وتكتب رسالة ، وتفصل الحسابات .

والآن دعنا نعبر عن هذا التخطيط بالشفرة الشبيهة .

```
PROGRAM depreciation1(input,output);

(* variable declarations *)

BEGIN

(* initialize flag *)

REPEAT

   (* read choice *)

   IF choice <> 4 THEN (* read val and n *)

   CASE choice OF
      1 : BEGIN

            (* write heading: Straight-Line Method *)

            deprec := val/n;
            FOR year := 1 TO n DO
               BEGIN
                  val := val-deprec;

                  (* write year, deprec, val *)

               END
```

(تكملة البرنامج فى الصفحة التالية)

```
        END;
    2 : BEGIN

           (* write heading: Double Declining Balance Method *)

           FOR year := 1 TO n DO
              BEGIN
                 deprec := 2*val/n;
                 val := val-deprec;

                 (* write year, deprec, val *)

              END
        END;
    3 : BEGIN

           (* write heading: Sum-of-the-Years'-Digits Method *)

           tag := val;
           FOR year := 1 TO n DO
              BEGIN
                 deprec := (n-year+1)*tag/(n*(n+1)/2);
                 val := val-deprec;

                 (* write year, deprec, val *)

              END
        END;
    4 : BEGIN

           (* write message *)

           flag := false;
        END
  END
UNTIL flag = false
END.
```

ومعظم الشفرة الشبيهة مباشرة ، بالرغم من وجود بعض التعليقات خلالها . أولا نرى أن مكون REPEAT-UNTIL مستخدم لتكرار مجموعة من الحسابات . وداخل هذا المكون استخدام مكون الحالة CASE لاختيار طريقة استهلاك محددة . وكل طريقة استهلاك تستخدم مكون FOR لعمل دورة خلال عمر العنصر الافتراضي ، والمقدر بأنه ( n-year ) n من السنوات .

ويبدو حساب الاستهلاك باستخدام طريقة مجموعة الأرقام الدالة على السنين بهما . والاصطلاح ( n - year + 1 ) يحتاج بصفة خاصة إلى توضيح . وتستخدم هذه الكمية في العد التنازلي من n وحتى 1 مع تقدم السنين للأمام ( من 1 إلى n ) . وهذه القيم المتناقصة مطلوبة في طريقة مجموعة الأرقام الدالة على السنين . وقد كان يمكننا بالطبع أن نعد دورة عد تنازلية بدلا من ذلك ( أي يمكن استخدام FOR year := n DOWNTO 1 DO ) ، إلا أن ذلك يتطلب كتابة دورة عد تصاعدية مناظرة لإخراج الاستهلاك السنوي لجميع السنين في الترتيب المناسب .

وعند هذه النقطة لايكون من الصعب كتابة برنامج بسكال كاملا على النحو التالى :

```
PROGRAM depreciation1(input,output);

(* THIS PROGRAM CALCULATES DEPRECIATION INTERACTIVELY,
   USING ONE OF THREE POSSIBLE METHODS *)

VAR n,choice,year : integer;
    val,deprec,tag : real;
    flag : boolean;

BEGIN   (* action statements *)
   flag := true;
   REPEAT
      page;      (* Begin input routine *)
      write(' Method: (1-SL  2-DDB  3-SYD  4-End) ');
      readln(choice);
      IF choice <> 4 THEN
      BEGIN
         write(' Original value: ');
         readln(val);
         write(' Number of years: ');
         readln(n);
         writeln
      END;

      CASE choice OF

         1 : BEGIN   (* SL *)
                writeln(' Straight-Line Method');
                writeln;
                deprec := val/n;
                FOR year := 1 TO n DO
                   BEGIN
                      val := val-deprec;
                      write(' End of Year ',year:2);
                      write('   Depreciation: ',deprec:5:0);
                      writeln('   Current Value: ',val:6:0)
                   END
             END;

         2 : BEGIN   (* DDB *)
                writeln(' Double Declining Balance Method');
                writeln;
                FOR year := 1 TO n DO


                   BEGIN
                      deprec := 2*val/n;
                      val := val-deprec;
                      write(' End of Year ',year:2);
                      write('   Depreciation: ',deprec:5:0);
                      writeln('   Current Value: ',val:6:0)
                   END
             END;
```

(تكملة البرنامج فى الصفحة التالية)

```
       3 : BEGIN   (* SYD *)
             writeln(' Sum-of-the-Years''-Digits Method');
             writeln;
             tag := val;
             FOR year := 1 TO n DO
                BEGIN
                   deprec := (n-year+1)*tag/(n*(n+1)/2);
                   val := val-deprec;
                   write(' End of Year ',year:2);
                   write('   Depreciation: ',deprec:5:0);
                   writeln('   Current Value: ',val:6:0)
                END
          END;

     4 : BEGIN   (* end computation *)
             writeln(' That''s all, folks!');
             flag := false;
          END
    END   (* choice *)
  UNTIL flag = false
END.
```

وقد صمم البرنامج للعمل المتداخل ؛ مع وجود ملقنات لبيانات المدخلات المطلوبة . لاحظ أن البرنامج ينتج قائمة بها أربعة اختيارات لحساب الاستهلاك باستخدام إحدى الطرق الثلاث ، أو لإنهاء الحسابات . وسوف يستمر الكمبيوتر فى قبول مجموعات جديدة من بيانات المدخلات ، وأداء الحسابات المناسبة لكل مجموعة من مجموعة البيانات ، حتى يتم اختيار المقدار 4 من القائمة .

وموضح أدناه بعض المخرجات ، حيث وضع خط تحت استجابات المستفيد . وفى كل حالة يحدث استهلاك لعنصر قيمته الأصلية 8000 دولار ، وذلك على عشر سنوات باستخدام كل طريقة من الطرق الثلاث . وأخيرا تفصل الحسابات كاستجابة لآخر اختيار فى القائمة .

```
Method: (1-SL  2-DDB  3-SYD  4-End) 1
Original value: 8000
Number of years: 10

Straight-Line Method

End of Year  1   Depreciation:  800.   Current Value:  7200.
End of Year  2   Depreciation:  800.   Current Value:  6400.
End of Year  3   Depreciation:  800.   Current Value:  5600.
End of Year  4   Depreciation:  800.   Current Value:  4800.
End of Year  5   Depreciation:  800.   Current Value:  4000.
End of Year  6   Depreciation:  800.   Current Value:  3200.
End of Year  7   Depreciation:  800.   Current Value:  2400.
End of Year  8   Depreciation:  800.   Current Value:  1600.
End of Year  9   Depreciation:  800.   Current Value:   800.
End of Year 10   Depreciation:  800.   Current Value:     0.

Method: (1-SL  2-DDB  3-SYD  4-End) 2
Original value: 8000
Number of years: 10

Double Declining Balance Method

End of Year  1   Depreciation: 1600.   Current Value:  6400.
End of Year  2   Depreciation: 1280.   Current Value:  5120.
End of Year  3   Depreciation: 1024.   Current Value:  4096.
```

(تكملة البرنامج فى الصفحة التالية)

```
End of Year  4   Depreciation:  819.   Current Value:  3277.
End of Year  5   Depreciation:  655.   Current Value:  2621.
End of Year  6   Depreciation:  524.   Current Value:  2097.
End of Year  7   Depreciation:  419.   Current Value:  1678.
End of Year  8   Depreciation:  336.   Current Value:  1342.
End or Year  9   Depreciation:  268.   Current Value:  1074.
End of Year 10   Depreciation:  215.   Current Value:   859.


Method: (1-SL  2-DDB  3-SYD  4-End) 3
Original value: 8000
Number of years: 10

Sum-of-the-Years'-Digits Method

End of Year  1   Depreciation: 1455.   Current Value:  6545.
End of Year  2   Depreciation: 1309.   Current Value:  5236.
End of Year  3   Depreciation: 1164.   Current Value:  4073.
End of Year  4   Depreciation: 1018.   Current Value:  3055.
End of Year  5   Depreciation:  873.   Current Value:  2182.
End of Year  6   Depreciation:  727.   Current Value:  1455.
End of Year  7   Depreciation:  582.   Current Value:   873.
End of Year  8   Depreciation:  436.   Current Value:   436.
End of Year  9   Depreciation:  291.   Current Value:   145.
End of Year 10   Depreciation:  145.   Current Value:     0.


Method: (1-SL  2-DDB  3-SYD  4-End) 4
That's all, folks!
```

لاحظ أن آخر طريقتين ينتج عنهما استهلاك سنوى كبير خلال السنوات الأولى ، واستهلاك سنوى صغير جدا فى آخر سنوات العمر الافتراضى للعنصر ، كما نرى أيضا أن قيمة العنصر تصبح صفرا فى نهاية عمره الافتراضى عند استخدام كل من الطريقة الأولى والطريقة الثالثة ، إلا أنه عند استخدام الطريقة الثانية تبقى قيمة صغيرة غير مستهلكة عند انتهاء عمر العنصر الافتراضى .

## ٨ – عبارة اذهب إلى : 8. THE GOTO STATEMENT

عبارة اذهب إلى GOTO هى عبارة بسيطة تستخدم لإبدال تسلسل تنفيذ البرنامج ، وذلك بنقل التحكم ( أى بعمل قفزات ) إلى جزء بعيد من البرنامج . وصيغة عبارة GOTO هى :

```
GOTO statement label
```

حيث يمثل عنوان العبارة statement label رقما موجبا ، لايزيد عن 9999 .

### مثال (٦-١٨)

فيما يلى عدة عبارات GOTO تقليدية :

```
GOTO 100;

IF flag THEN GOTO 9999;

IF sum > 100.0 THEN BEGIN
                      writeln(sum);
                      GOTO 35
                    END;
```

ويجب توضيح كل عناوين العبارات قبل إمكانية استخدامها داخل البرنامج . ( تذكر أن توضيح العناوين label declaration يجب أن يسبق توضيحات الثوابت والمتغيرات ، كما سبق ذكره فى قسم ٥من الفصل الأول ، وقسم ٢ من الفصل الخامس ) . وتأخذ توضيحات العناوين الصورة التالية :

حيث statement 1 label ، و statement 2 label .. تمثل عناوين فردية للعبارات . ( لاحظ التمييز بين عناوين العبارات statement labels وعناوين الحالة Case labels التى سبق ذكرها فى القسم السابق . لاحظ بصفة خاصة أن عناوين العبارات مقيدة ، بحيث أنها يجب أن تكون أرقاما موجبة ، بينما عناوين الحالة ليست مقيدة . كما أن عناوين العبارات يجب توضيحها قبل أن يمكن استخدامها ، بينما لايتم أى توضيح لعناوين الحالة ) .

**مثـــال (٦-١٩)**

فيما يلى جزءًا من برنامج بسكال يحتوى على توضيح عناوين :

```
PROGRAM sample(input,output);
LABEL 100,200,300;
CONST factor = 0.5;
VAR a,b,c : real;
```

لاحظ أن توضيح العناوين يلى عبارة PROGRAM ، ويسبق تعريف الثوابت ، طبقا لما هو مطلوب .

وتكتب العبارات التى لها عناوين على النحو التالى :

*statement label : statement*

وقد تسبق مثل هذه العبارات عبارات GOTO المناظرة لها ، أو تليها .

**مـــثال (٦-٢٠)**

فيما يلى عدة عبارات بسكال . كل منها يحتوى على عنوان لعبارة

```
10 : readln(a,b,c);

20 : FOR count := 1 TO n DO
       BEGIN
         .
         .
         .
       END;

99 : BEGIN
       writeln(sum);
       IF flag THEN writeln(x,y,z);
       writeln
     END;
```

ويتسبب تنفيذ عبارة GOTO فى نقل التحكم إلى العبارات المناظرة لهذه العناوين . وعلى هذا ... فتكون العبارة التى لها العنوان هى التى تنفذ بعد GOTO .

ويمكن أن يحتوى البرنامج على عدة عبارات GOTO تنقل التحكم إلى نفس الموقع ( إلى نفس العبارة البعيدة ) داخل البرنامج . وعلى أية حال ... يجب أن يكون لكل عبارة معنونة عنوان خاص بها ( أى أنه لايمكن أن تأخذ عبارتان مختلفان نفس العنوان ) .

مثـال (٦-٢١)

فيما يلى جزءًا من برنامج بسكال :

```
PROGRAM sample(input,output);
LABEL 10,20;
CONST . . .;
VAR   . . .;
BEGIN
    .
    .
    .
   10 : readln(a,b,c);
            .
            .
            .
        IF a <= 0 THEN GOTO 20;
            .
            .
            .
        GOTO 10;
            .
            .
            .
        IF flag THEN GOTO 20;
            .
            .
            .
   20 : writeln(x,y,z);
    .
    .
    .
END.
```

لاحظ أنه يمكن نقل التحكم إلى العبارة رقم 20 من موقعين مختلفين داخل البرنامج . وكل من عملتى النقل شرطية ، أى أنها تعتمد على صحة أحد تعبيرات بوليان .

ويجب تجنب مثل هذا النوع من العبارات فى كتابة برنامج البسكال بصفة عامة . وفى واقع الأمر ... فإن العديد من سمات البسكال صممت خصيصا لمنع الحاجة لهذا المنطق فى البرنامج . ( سوف يذكر فيما بعد المزيد عن ذلك ) .

ويجب الحرص عند استخدام عبارة GOTO مع العبارات المركبة ، فيمكن نقل التحكم خارج out of أو داخل Within العبارة المركبة ، كما يمكن نقل التحكم إلى بداية beginning العبارة المركبة . ولايمكن على أية حال نقل التحكم إلى داخل into عبارة مركبة . والأكثر من هذا ... إذا مانقل التحكم داخليا إلى END الخاصة بعبارة مركبة ( أى إذا ماوضع عنوان للكلمة الرئيسية END ) ، فيجب أن يسبق العنوان عبارة فارغة . ويتحقق ذلك عن طريق وضع فاصلة منقوطة بعد العبارة التى تسبق END .

وهذه القيود موضحة فى الأمثلة التالية :

مـثال (٦-٢٢)

فيما يلى مثالا لنقل مسموح به للتحكم خارج عبارة مركبة .

```
        BEGIN
            .
            .
            .
            IF flag THEN GOTO 50;
            .
            .
            .
        END;
          .
          .
          .
 50 : writeln(answer);
```

**مـثال (٦-٢٣)**

فيما يلى مثالا لنقل مسموح به للتحكم داخل عبارة مركبة .

```
        BEGIN
            .
            .
            .
            IF flag THEN GOTO 20;
            .
            .
            .
       20 : writeln(answer);
            .
            .
            .
        END;
```

**مـثال (٦-٢٤)**

اعتبر الآن نقل التحكم التالى إلى END الخاصة بعبارة مركبة .

```
        BEGIN
            .
            .
            .
            IF flag THEN GOTO 30;
            .
            .
            .
            writeln(answer);
       30 : END;
```

لاحظ الفاصلة المنقوطة الموجودة بعد عبارة Writeln . ينتج عن ذلك عبارة فارغة بين Writeln و END كما هو مطلوب تماما .

### مـثال (٦-٢٥)

**اعتبر أخيرا المحاولة غير المسموح بها illegal التالية لنقل التحكم إلى داخل عبارة مركبة .**

```
IF flag.THEN GOTO 40;
  .
  .
  .
  BEGIN
     .
     .
     .
  40 : writeln(answer);
     .
     .
     .
  END;
```

**إذا ماالحتوى أحد البرامج مثل هذا التكوين ، فينتج عن ذلك خطأ تكوينى أثناء عملية الترجمة .**

وهناك قيود شبيهة على استخدام عبارة GOTO مع مكونات التحكم الأخرى ، فيمكن نقل التحكم داخل مكون التحكم أو خارجه ، ولايمكن نقله إلى داخله . وأكثر من ذلك ... لاتقبل بعض مترجمات البسكال وجود عبارات معنونة داخل مكونات WHILE و FOR و IF . وعلى هذا ... فمن المفضل تجنب استخدام عبارات GOTO في مثل هذه الحالات .

كما أنه هناك قيود أيضا على استخدام عبارة GOTO التى تنتمى إلى مدى البرنامج program scope . وسوف يناقش هذا الأمر فى الفصل القادم . وفى كلمات عامة سوف نرى أنه يمكن إجراء تداخل للمجموعات blocks كما يحدث ذلك فى مكونات التحكم الموجودة فى البرنامج . وفى مثل هذه الحالات يمكن نقل التحكم داخل المجموعة block ، ومن خارج المجموعة إلى مجموعة مشمولة enclosing ، ولايمكن نقل التحكم إلى داخل مجموعة متوازنة .

وأخيرا يجب تمييز أن استخدام عبارات GOTO غير مرغوب فيه فى البسكال ، حيث إنه يبدل من المسار التتابعى للمنطق ، ويقلل من الوضوح ، وهذا عكس ماتتميز به هذه اللغة . وفى واقع الأمر هناك بعض علماء الكمبيوتر الذين يضعون حظرا كاملا على استخدام عبارات GOTO ، بالرغم من أن هذا يعتبر تطرفا بعض الشئ . وهناك اتفاق شبه عام على أنه يجب تجنب استخدام GOTO بقدر الإمكان .

وعمليا يمكن أن توجد عبارة GOTO داخل مكون IF ، وذلك لنقل التحكم ( شرطـيا ) إلـى نهايـة البرنامـج . ( تفضل مكونات WHILE-DO و REPEAT - UNTIL فى المواقف التى تتطلب نقل التحكم إلى بداية البرنامج ) . وعادة ماتحدث هذه التنقلات بالاتصال مع بعض أحد أنواع الاستراتيجيات العامة global ( أى القفز إلى نهاية البرنامج إذا ماكانت إحدى قيم المدخلات سالبة ) . وفى الناحية الأخرى يجب ألا يحتوى البرنامج تحت أى ظروف على عدد كبير من القفزات المحلية localized ، فمثل هذه البرامج تتنافى مع تصميم لغة البسكال الذى يهدف لوضوح منطق البرنامج .

### مـثال (٦-٢٦)

حساب قيمة متوسط قائمة من الأعداد . دعنا نعتبر مرة أخرى مشكلة حساب متوسط قائمة من الأعداد ، كما فى مثال ٦ - ٤ ، ومثال ٦ - ٥ ، ومثال ٦ - ٨ . افترض الآن أن طول القائمة غير معروف مسبقاً وبدلا من ذلك علينا أن نستمر فى قراءة وجمع كميات مدخلات متتالية ، حتى تقرأ قيمة سالبة فى الكمبيوتر . وهذه القيمة السالبة تفسر بأنها معيار للتوقف ، ولاتدخل فى حساب المتوسط .

وتجرى الحسابات على النحو التالى :

(١) تحديد قيمة 0 للمتغير الصحيح count .

(٢) تحديد قيمة 0 للمتغير الحقيقى sum .

(٣) تحديد قيمة صحيح للمتغير البوليان flag .

(٤) تكرار أداء الخطوات التالية ، طالما أن قيمة flag صحيحة true .

(أ) قراءة أحد الأعداد من القائمة ( كل عدد ممثل بمتغير حقيقى x ) .

(ب) إذا كانت قيمة x سالبة يتجه التحكم GOTO إلى الخطوات رقم 5 ، وإلا فتضاف القيمة x إلى sum ، وتزداد قيمة count بمقدار 1 .

(٥) قسمة sum على count للحصول على قيمة المتوسط .

(٦) طباعة أو كتابة قيم count أو average .

وفيما يلى برنامج البسكال المناظر لذلك .

```
PROGRAM average5(input,output);

(* THIS PROGRAM CALCULATES THE AVERAGE OF A LIST OF NUMBERS .
   USING THE WHILE - DO STRUCTURE AND A GOTO STATEMENT *)

LABEL 10;

VAR count : integer;
    x,sum,average : real;
    flag : boolean;

BEGIN   (* action statements *)
     count := 0;
     sum := 0;
     flag := true;
     WHILE flag DO
        BEGIN
           readln(x);
           IF x < 0 THEN GOTO 10;
           sum := sum+x;
           count := count+1
        END;
10 : average := sum/count;
     writeln(' The average of ',count:5,' numbers is ',average)
END.
```

لاحظ أن قيمة flag تظل صحيحة true خلال هذا البرنامج . وعلى هذا ... فيستمر التكرار دون توقف حتى تظهر قيمة سالبة للمتغير x ، ويتم إدخالها داخل الكمبيوتر . وبمجرد اكتشاف قيمة سالبة للمتغير x ، ينقل التحكم خارج الدورة إلى العبارة رقم 10 . عند ذلك تسحب قيمة المتوسط ، وتكتب كمخرجات .

ويجب أن يحاول القارئ أن يعد هذا البرنامج بنفسه بدون استخدام عبارة GOTO ( انظر مشكلة ٤٥ فى نهاية هذا الفصل ) . ويمكن كتابة مثل هذا البرنامج ، إلا أنه أكثر إرهاقا عن البرنامج المستخدم لعبارة GOTO .

## أسئلة للمراجعة : Review Questions

(١) ماذا يعنى التكرار repetition ؟

(٢) ماهى الدورة الشرطية ؟ وماهى الدورة غير الشرطية ؟ ماهى الاختلافات بين الدورة الشرطية ، والدورة غير الشـرطية ؟

(٣) ماذا يعنى التنفيذ الشرطى ؟

(٤) ماذا يعنى الاختيار ؟

(٥) ماهى القيم التى يمكن أن تمثل بواسطة تعبير بوليان ؟

(٦) لخص القواعد المصاحبة لتعبيرات بوليان .

(٧) لخص الفروق بين المؤثرات العلاقية والمؤثرات المنطقية . مانوع العناصر المستخدمة مع كل من هذين النوعين من المؤثرات ؟

(٨) ماهو الفرق بين العبارات البسيطة والعبارات المركبة ؟

(٩) اذكر ثلاثة أنواع من العبارات البسيطة .

(١٠) لخص القواعد التكوينية التى تصاحب العبارات المركبة .

(١١) ماهو الفرض من مكون WHILE-DO ؟ ومتى يتم تقويم تعبير بوليان ؟ ماهو أقل عدد من المرات ينفذ فيه مكون WHILE-DO ؟

(١٢) كيف ينتهى تنفيذ مكون WHILE-DO ؟

(١٣) لخص القواعد التكوينية المصاحبة لمكون WHILE-DO.

(١٤) ماهو الفرض من مكون REPEAT-UNTIL ؟ وكيف يختلف عن مكون WHILE-DO ؟

(١٥) ماهو أقل عدد من المرات ينفذ فيه مكون REPEAT-UNTIL ؟ قارن ذلك مع مكون WHILE-DO ، ووضح سبب هذه الاختلافات .

(١٦) لخص القواعد التكوينية المصاحبة لمكون REPEAT-UNTIL . قارن ذلك بمكون WHILE-DO .

(١٧) ماهو الفرض من مكون FOR ؟ وكيف يختلف هذا المكون عن مكون WHILE-DO وعن مكون REPEAT-UNTIL ؟

(١٨) كم عدد مرات تنفيذ مكون FOR ؟ قارن ذلك بمكون WHILE-DO ، ومكون REPEAT-UNTIL .

(١٩) ماهو الفرض من متغير التحكم الموجود فى مكون FOR ؟ ومانوع المتغير الذى يمكن استخدامه ؟

(٢٠) صف الصيغتين المختلفتين لمكون FOR . ماهو الفرض من كل منهما ؟

(٢١) لخص القواعد التكوينية المصاحبة لمكون FOR .

(٢٢) ماهى القواعد المستخدمة فى عمل مكونات تحكم متداخلة ؟ هل يمكن أن يحتوى أحد أنواع مكونات التحكم نوعا آخر ؟

(٢٣) ماهو الغرض من مكون IF ؟ وكيف يختلف هذا المكون عن مكون WHILE-DO ، ومكون REPEAT-UNTIL ومكون FOR ؟

(٢٤) صف صيغتين مختلفتين لمكون IF . وكيف تختلف هاتان الصيغتان عن بعضهما ؟

(٢٥) لخص القواعد التكوينية المصاحبة لمكون IF.

(٢٦) كيف تفسر مكونات IF المتداخلة ؟ وبصفة خاصة كيف يفسر المكون التالى :

(٢٧) ماهو الغرض من مكون CASE ؟ وكيف يختلف هذا المكون عن المكونات الأخرى المذكورة فى هذا الفصل ؟

(٢٨) ماهو القائم بالاختيار ؟ وماهو الغرض الذى يوجد من أجله ؟

(٢٩) ماهى القيم التى يمكن تحديدها لعناوين الحالة ؟

(٣٠) ماذا يحدث عندما تكون قيمة القائم بالاختيار هى نفسها قيمة أحد العناوين ؟ وماذا يحدث عندما لاتكون قيمة القائم بالاختيار هى نفسها قيمة أى عنوان من عناوين الحالة ؟

(٣١) لخص القواعد التكوينية المصاحبة لتكوين CASE . هل يمكن أن يصاحب عناوين عبارة واحدة ؟

(٣٢) قارن مكون CASE مع استخدام مكونات IF - THEN - ELSE المتداخلة . أى منها أكثر إقناعا ؟

(٣٣) ماهو الغرض من عبارة GOTO ؟ وهل هى عبارة بسيطة أم عبارة مركبة ؟

(٣٤) لخص القواعد التكوينية التى تصاحب عناوين العبارات . ماهى القيم التى يمكن استخدامها كعناوين للعبارات ؟ وكيف يمكن توضيحها ؟ كيف يصاحب عنوان العبارة عبارة بعيدة ؟

(٣٥) قارن التكوين المصاحب لعناوين العبارات ، مع التكوين المصاحب لعبارات الحالة . لاحظ الفروق .

(٣٦) صف القيود التى تقع على استخدام عبارة GOTO مع استخدام عبارات مركبة . وهل تطبق نفس القيود عند استخدام عبارة GOTO مع مكونات تحكم أخرى ؟

(٣٧) لماذا لايشجع استخدام عبارات GOTO فى البسكال ؟ وتحت أى ظروف يمكن أن تكون عبارات GOTO مفيدة ؟ مانوع الاستخدام الذى يجب تجنبه ؟ ولماذا ؟ ناقش ذلك بالتفصيل .

## Solved Problems — مشاكل محلولة :

(٣٨) حدد قيمة كل تعبير بوليان من التعبيرات التالية ، مستخدما المعرفات المعرفة أدناه :

```
CONST a = 50;
      t = 0.02;
      x = 'P';
```

| Boolean Expression | Value |
|---|---|
| `a < 12` | false |
| `abs(t) = a/2500` | appears true, but probably false because of numerical inaccuracies |
| `x <> 'p'` | true |
| `x = a` | error—mixed data types |
| `(a > 6) OR (t <= 0)` | true |
| `(a > 6) AND (t <= 0)` | false |
| `((x = 'P') AND (t = 0)) OR (x > 'A')` | true |
| `(t >= 0) AND (NOT (x <> 'P'))` | true |

(٣٩) كل من هيكل مكون التحكم التالية يوضح طريقة مختلفة لتنفيذ عملية دوره 100 مرة . ( افرض أن count متغير صحيح ) .

(*a*)
```
count := 1;
WHILE count <= 100 DO
   BEGIN
      .
      .
      .
      count := count + 1
   END;
```

(*b*)
```
count := 1;
REPEAT
   .
   .
   .
   count := count + 1
UNTIL count > 100;
```

(*c*)
```
FOR count := 1 TO 100 DO
   BEGIN
      .
      .
      .
   END;
```

(*d*)
```
FOR count := 100 DOWNTO 1 DO
   BEGIN
      .
      .
      .
   END;
```

(٤٠) كل من التخطيط التالى يوضح طريقة مختلفة لمكونات تحكم متداخلة :

```
(a)  VAR i : integer;
         flag : boolean;
        .
        .
        .
     flag := true;
     WHILE flag DO
        BEGIN
           FOR i := 1 TO n DO
              BEGIN
                 .
                 .
                 .
              END;
              .
              .
              .
              IF . . . THEN flag := false
        END;

(b)  VAR i : integer;
         flag : boolean;
        .
        .
        .
     flag := true;
     REPEAT
        .
        .
        .
        IF . . . THEN FOR i := 1 TO n DO
                         BEGIN
                            .
                            .
                            .
                         END
                      ELSE FOR i := n DOWNTO 1 DO
                         BEGIN
                            .
                            .
                            .
                         END;
        IF . . . THEN flag := false
     UNTIL NOT flag;

(c)  VAR c : char;
         count : integer;
        .
        .
        .
     IF c <> '*'
```

```
        THEN REPEAT
                .
                .
                .
             UNTIL c = '*'
        ELSE FOR count := 1 TO 3 DO BEGIN
                                       .
                                       .
                                       .
                                   END;
```

(*d*)
```
VAR count,start,finish,test,i : integer;
   .
   .
   .
FOR count := start TO finish DO
   BEGIN
      .
      .
      .
      test := 1;
      WHILE test < 10 DO
         FOR i := 1 TO test DO
            BEGIN
               .
               .
               .
            END;
         test := test + 1;
            .
            .
            .
   END;
```

(٤١) يوضح التخطيط التالى طريقتين مختلفتين لاختيار أحد إجراءات ثلاثة ( افرض أن choice متغير صحيح ) .

(*a*)
```
IF choice =1 THEN . . .
             ELSE IF choice = 2 THEN . . .
                                 ELSE . . . ;
```

(*b*)
```
CASE choice OF
   1 : . . . ;
   2 : . . . ;
   3 : . . .
END;
```

لاحظ أن استخدام مكونات IF - THEN - ELSE المتداخلة كجزء من ( a ) يصبح أكثر أرهاقا كلما ازداد عدد الاختيارات .

## Supplementray Problems مشاكل متكاملة :

(٤٢) حدد قيمة كل تعبير بوليان من التعبيرات التالية ، مستخدما المعرفات أدناه :

```
CONST f = 300;
      p = -0.001;
      q = 0.001;
      c = '5';
```

(*a*) `2*f >= 500`

(*b*) `abs(p) = abs(q)`

(*c*) `c = 5`

(*d*) `p + q > 0`

(*e*) `(abs(p) = q) AND (c > '4')`

(*f*) `(p = abs(q)) OR (c > 4)`

(*g*) `sqr(p) < sqrt(q)`

(*h*) `(q < 0) OR ((f > 0) AND (f < 100))`

(*i*) `NOT (c < '7')`

(٤٣) فيما يلى تخطيطا لعدة تكوينات تحكم ، وبعضها مكتوب بطريقة خاطئة . حدد كل الأخطاء :

(*a*)
```
VAR result : real;
   .
   .
   .
WHILE result > 0 DO
   BEGIN
      .
      .
      .
   END;
```

(*b*)
```
VAR a,b,c,d : integer;
   .
   .
   .
REPEAT
   .
   .
   .
UNTIL (a+b) > (2*c-d);
```

(*c*)
```
VAR x,n1,n2 : real;
   .
   .
   .
FOR x := n1 TO n2 DO
   BEGIN
      .
      .
      .
   END;
```

(*d*)
```
VAR x : char;
   .
   .
   .
FOR x := 'z' DOWNTO 'a' DO
   BEGIN
      .
      .
      .
   END;
```

(*e*)
```
VAR a,b,c : integer;
   .
   .
   .
FOR a := b TO c DO
   BEGIN
      .
      .
      .
   END;
```

(*f*)
```
VAR a,b,c,d : integer;
   .
   .
   .
FOR a := b DOWNTO d DO
   BEGIN
      .
      .
      .
      FOR c := b TO d DO
         BEGIN
            .
            .
            .
         END
   END;
```

(*g*)
```
VAR a,b,c,d : integer;
   .
   .
   .
FOR b := a TO c DO
   BEGIN
      .
      .
      .
      FOR d := a TO b DO
         BEGIN
            .
            .
            .
         END
   END;
```

(*h*)
```
VAR flag : boolean;
    c : char;
   .
   .
   .
flag := true;
WHILE flag DO
   BEGIN
      .
      .
      .
      REPEAT
         .
         .
         read(c);
         IF c = '*' THEN flag := false;
         .
         .
      UNTIL NOT flag
   END;
```

(*i*)
```
VAR i,j : integer;
    x : real;
    flag : boolean;
   .
   .
   .
IF i < 0 THEN IF j > 10 THEN flag := true
                        ELSE flag := false ELSE x := -0.1;
```

(*j*)
```
VAR x,y : real;
   .
   .
   .
CASE sqr(x) OF
     1.0 : y := x;
     4.0 : y := 2*x;
     9.0 : y := 3*x;
    16.0 : y := 4*x
END;
```

(*k*)
```
LABEL 10;
VAR c : char;
   .
   .
   .
read(c);
IF c = '*' THEN GOTO 10;
   .
   .
   .
WHILE c <> '*' DO
   BEGIN
      .
      .
      .
 10 : write(c);
      .
      .
      .
   END;
```

(*l*)
```
LABEL 1,2,3;
VAR i : integer;
   .
   .
   .
CASE i OF
   1 : BEGIN . . . GOTO 3 . . . END;
   2 : BEGIN . . . GOTO 3 . . . END;
   3 : BEGIN . . . END
END;
```

## مشاكل برمجة : Programming Problems

(٤٤) ترجم ونفذ البرامج المعطاه فى الأمثلة ٦-٤ ، و٦-٥ ، و٦-٨ ، مستخدما العشرة أرقام التالية :

| | |
|---|---|
| 27.5 | 87.0 |
| 13.4 | 39.9 |
| 53.8 | 47.7 |
| 29.2 | 8.1 |
| 74.5 | 63.2 |

(٤٥) عدل البرنامج المعطى فى مثال ٦ -٢٦، والذى يحسب متوسط قائمة أعداد غير معرف طولها ، مع حذف عبارة GOTO . اكتب برنامجا جديدا بطريقتين مختلفتين :

(أ) باستخدام مكون WHILE-DO .

(ب) باستخدام مكون REPEAT - UNTIL .

اختبر كل صيغة للبرنامج ، مستخدما بيانات المشكلة السابقة .

(٤٦) أعد كتابة برنامج الاستهلاك المعطى فى مثال ٦ -١٧لاستخدام مكون IF - THEN - ELSE ، بدلا من مكون CASE . اختبر البرنامج مستخدما البيانات المعطاه فى مثال ٦ -١٧. أى صيغة من هاتين الصيغتين تفضلها ؟ وماسبب ذلك؟

(٤٧) المعادلة :

$$x^5 + 3x^2 - 10 = 0$$

والمقدمة فى مثال ٦ -١٤يمكن إعادة ترتيبها لتأخذ الصيغة :

$$x = [(10 - x^5)/3]^{1/2}$$

أعد كتابة برنامج البسكال المقدم فى مثال ٦ -١٤لاستخدام الصيغة المذكورة أعلاه للمعادلة . نفذ البرنامج ، وقارن النتائج المحسوبة مع النتائج المعطاه فى مثال ٦ -١٤. لماذا تختلف النتائج ؟ هل تخطئ أجهزة الكمبيوتر ؟

(٤٨) عدل البرنامج المعطى فى مثال ٦ -١٤، والذى يحل معادلة جبرية مع إحلال مكون WHILE-DO بمكون REPEAT - UNTIL . أى المكونين يناسب هذه المشكلة الخاصة أفضل ؟

(٤٩) عدل البرنامج المعطى فى مثال ٦ -١٤، والذى يحل معادلة جبرية مع إحلال مكون FOR - TO محل مكون WHILE-DO . قارن استخدام مكون FOR - TO مع مكون WHILE-DO ، مع مكون REPEAT - UNTIL . أى هذه المكونات تفضل ؟ ولماذا ؟

(٥٠) اكتب برنامجا كاملا بلغة البسكال لكل مشكلة من المشاكل التالية . استخدام مكون التحكم الأكثر ملائمة لكل مشكلة . ابدأ بتخطيط تفصيلى ثم أعد كتابة هذا التخطيط بالشفرة الشبيهة إذا لم يكن التحويل المباشر إلى برنامج البسكال غير واضح . تأكد من استخدام نظام برمجة جيد ( مع وجود تعليقات وترحيلات وخلافه ) .

(أ) احسب المتوسط المرجح weighted average لقائمة من n عدد مستخدما العلاقة .

$$x_{avg} = f_1x_1 + f_2x_2 + \cdots + f_nx_n$$

حيث f1 و f2 ، ... هى عوامل ترجيح weighting factors كسرية ، بحيث إن :

$$0 <= f_i < 1 \qquad \text{and} \qquad f_1 + f_2 + \cdots + f_n = 1$$

اختبر برنامجك بالبيانات التالية :

| $i$ | $f$ | $x$ |
|---|---|---|
| 1 | 0.06 | 27.5 |
| 2 | 0.08 | 13.4 |
| 3 | 0.08 | 53.8 |
| 4 | 0.10 | 29.2 |
| 5 | 0.10 | 74.5 |
| 6 | 0.10 | 87.0 |
| 7 | 0.12 | 39.9 |
| 8 | 0.12 | 47.7 |
| 9 | 0.12 | 8.1 |
| 10 | 0.12 | 63.2 |

(ب) احسب الضرب التراكمى لقائمة من n عدد . اختبر برنامجك ، مستخدما مجموعة البيانات التالية :

$(n = 6)$: 6.2, 12.3, 5.0, 18.8, 7.1, 12.8

(ج) أحسب الوسط الهندسى لقائمة أعداد مستخدما العلاقة التالية : $x_{avg} = [x_1 x_2 x_3 \cdots x_n]^{1/n}$

اختبر برنامجك مستخدما البيانات المعطاه فى الجزء ( b ) من المشكلة . قارن النتائج التى تحصل عليها بنتائج المتوسط الحسابى لنفس البيانات . أى متوسط منها هو الأكبر ؟

(د) حدد جذور المعادلة التالية :

$$ax^2 + bx + c = 0$$

مستخدما الصيغة المعروفة التالية :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

( انظر مثال ٨ – ٥ ) . خذ فى الاعتبار إمكانية أن يكون أحد الثوابت صفرا ، وأن الكمية $b^2 - 4ac$ تكون أقل من أو تساوى صفرا . اختبر البرنامج مستخدما مجموعة البيانات التالية :

| $a$ | $b$ | $c$ |
|---|---|---|
| 2 | 6 | 1 |
| 3 | 3 | 0 |
| 1 | 3 | 1 |
| 0 | 12 | −3 |
| 3 | 6 | 3 |
| 2 | −4 | 3 |

(هـ) أعداد فيبوناكى Fibonacci numbers لها تسلسل يكون فيه كل عدد مساويا مجموع العددين السابقين له . وفى كلمات أخرى ...

$$F_i = F_{i-1} + F_{i-2}$$

حيث تشير Fi إلى العدد i . وأول عددين طبقا للتعريف يساويان 1 ، أى أن $F_1 = F_2 = 1$

ملاحظة :

$$F_3 = F_2 + F_1 = 1 + 1 = 2$$
$$F_4 = F_3 + F_2 = 2 + 1 = 3$$
$$F_5 = F_4 + F_3 = 3 + 2 = 5$$

وهكذا :

اكتب برنامجا يحدد أول n عدد من أعداد فيبوناكى . اختبر البرنامج بقيمة n مساوية 23

(و) العدد الأولى Prime number هو رقم صحيح موجب ، يمكن قسمته على 1 أو على نفسه فقط بدون باق . احسب واعمل جدولاً من أول n عدد أولى . ( ملاحظة : العدد n يكون عددا أوليا إذا لم يكن خارج قسمة n/2 , n/3,..,n/sqrt(n) عبارة عن رقم صحيح ) . اختبر برنامجك بحساب أول 100 عدد أولى .

(ز) اكتب برنامجا على هيئة حوار يقرأ قيم أرقام صحيحة موجبة ، ويحدد مايلى :

١) إذا كان الرقم الصحيح عددا أوليا .

٢) إذا كان الرقم الصحيح عددا فيبوناكى .

اكتب برنامجا بطريقة يمكن أن يتكرر بها تنفيذ دورة ، حتى تكتشف قيمة صفر ككمية مدخلة . اختبر البرنامج بعدة قيم صحيحة من اختيارك .

(ح) احسب مجموعة أول n رقما فرديا ( أى 1+3+...+2*n-1 ) . اختبر البرنامج بحساب أول 100 رقم فردى ( ملاحظة أن آخر رقم هو 199 ) .

(ط) يمكن حساب جيب x بالتقريب بتجميع أول n حد من السلسلة النهائية .

$$\sin x = x - x^3/3! + x^5/5! - x^7/7! + \cdots$$

حيث n يعبر عنه بالتقدير الدائرى . اكتب برنامجا بلغة البسكال يقرأ قيمة x ، ثم يحسب جيبها . اكتب البرنامج بطريقتين مختلفتين .

١) جمع أول n حدا ، حيث n هى رقم صحيح يقرأ داخل الكمبيوتر مع القيمة العددية للزاوية x .

٢) استمر فى إضافة حدود متتالية فى السلسلة ، حتى تصبح قيمة الحد أقل من $10^{-5}$ ( فى قيمتها ) .

اختبر البرنامج لقيم x=1 و x=2 و x=3 . وفى كل حالة اكتب كمخرجات عدد الحدود المستخدمة فى الحصول على النتيجة النهائية .

(ى) افرض أن P من الدولارات استدينت من أحد البنوك ، على أن يعاد دفع A دولار كل شهر ، حتى يتم دفع القرض كله . وجزء من القسط الشهرى يمثل الفائدة محسوبة بأنها i% من القيمة الحالية المدان بها . وبقية القسط الشهرى تمثل جزءًا مدفوعًا من الدين . اكتب برنامجا بلغة البسكال يحدد المعلومات التالية :

١) قيمة الفائدة المدفوعة كل شهر .

٢) قيمة جزء الدين المسدد كل شهر .

٣) كمية الفائدة المتراكمة التى دفعت للبنك فى نهاية كل شهر .

٤) قيمة القرض التى مازال المدين مدينا بها .

٥) عدد الأقساط الشهرية اللازمة لإعادة الدين كاملا .

٦) قيمة آخر قسط ( حيث إنها ربما تكون أقل من A ) .

اختبر برنامجك مستخدما البيانات التالية :

$i = 1\ \%$ شهريا .

(ك) حصل مجموعة من الطلبة على الدرجات التالية فى ستة امتحانات فى لغة البسكال :

| Student Number | Exam Scores (percent) | | | | | |
|---|---|---|---|---|---|---|
| 10001 | 45 | 80 | 80 | 95 | 55 | 75 |
| 10002 | 60 | 50 | 70 | 75 | 55 | 80 |
| 10003 | 40 | 30 | 10 | 45 | 60 | 55 |
| 10004 | 0 | 5 | 5 | 0 | 10 | 5 |
| 10005 | 90 | 85 | 100 | 95 | 90 | 90 |
| 10006 | 95 | 90 | 80 | 95 | 85 | 80 |
| 10007 | 35 | 50 | 55 | 65 | 45 | 70 |
| 10008 | 75 | 60 | 75 | 60 | 70 | 80 |
| 10009 | 85 | 75 | 60 | 85 | 90 | 100 |
| 10010 | 50 | 60 | 50 | 35 | 65 | 70 |
| 10011 | 70 | 60 | 75 | 70 | 55 | 75 |
| 10012 | 10 | 25 | 35 | 20 | 30 | 10 |
| 10013 | 25 | 40 | 65 | 75 | 85 | 95 |
| 10014 | 65 | 80 | 70 | 100 | 60 | 95 |

اكتب برنامجاً تقليديا بلغة البسكال ، يقبل رقم كل طالب ودرجات الامتحانات كمدخلات ، ويحدد متوسط الدرجة لكل طالب ، ثم يخرج رقم الطالب مع درجات الاختبارات كلها والمتوسط المحسوب . اجعل البرنامج عاما كلما كان ذلك ممكنا .

(ل) عدل البرنامج المكتوب فى (ك) ليسمح بالترجيح غير المتساوى للدرجات الفردية للامتحانات . افرض بصفة خاصة أن كل امتحان من أول أربعة امتحانات يسهم بمقدار 15% من الدرجة النهائية ، وكل من الدرجتين الأخيرتين يسهم بمقدار 20% .

(م) وسع البرنامج الخاص بالجزء (ل) ، بحيث تحسب متوسط درجة الفصل كله بالإضافة إلى متوسطات الطلاب .

(ن) اكتب برنامجا بلغة البسكال ، يسمح باستخدام الكمبيوتر كحاسبة عادية . اعتبر العمليات الحسابية المعتادة فقط ( الجمع والطرح والضرب والقسمة ) . خذ فى الاعتبار ذاكرة يمكنها تخزين رقم واحد .

(س) انتج الهرم التالى من الأرقام ، مستخدما دورات متداخلة .

```
         1
        232
       34543
      4567654
     567898765
    67890109876
   7890123210987
  890123454321098
 90123456765432109
0123456789876543210
```

( لاتكتب ببساطة 10 سلاسل متعددة الأرقام )

(ع) انتج رسما لدالة .

$$y = e^{-0.1t} \sin (0.5t)$$

عن طريق طابع أسطر ، مستخدما نجمة (*) لكل نقطة تكون الرسم . اجعل الرسم يسرى رأسيا إلى أسفل الصفحة ، مع وجود نقطة واحدة ( نجمة واحدة ) فى السطر . ( ملاحظة : يجب أن يتكون كل سطر من نجمة واحدة ، يسبقها عدد مناسب من الفراغات . حدد موقع النجمة مستخدما دالة round ) .

(ف) اكتب برنامجا بلغة البسكال متداخلا يحول التاريخ الذى يتم إدخاله فى صورة mm-dd-yy ( مثال 69 - 12 - 4 ) إلى رقم يحدد عدد الأيام بعد أول يناير عام ١٩٦٠ . لعمل ذلك استخدم العلاقات التالية :

١) تاريخ السنة الحالية يمكن تحديده بطريقة تقريبية على النحو التالى :

```
day := trunc(30.42*(mm-1)) + dd
```

٢) إذا كانت mm = 2 ، أى أن الشهر هو فبراير ، أضف 1 إلى قيمة day .

٣) إذا كانت mm > 2 ، وكانت mm < 8 ( مارس وأبريل ومايو ويونيو ويوليو ) ، قلل قيمة day بمقدار 1 .

٤) إذا كان yy MOD = 0 و mm > 2 ( سنة كبيسة ) ؛ أضف 1 إلى قيمة day .

٥) أضف 1461 إلى day لكل دورة مكونة من 4 سنوات بعد 60-1-1 .

٦) أضف 365 إلى day لكل سنة كاملة بعد إمام آخر دورة طولها أقل من 4 سنوات ، ثم أضف 1 ( لأقرب سنة كبيسة ) .

اختبر البرنامج بتاريخ اليوم ، أو بأى تاريخ تختاره .

الفصل السابع

# الإجراءات والدوال

# Procedures and Functions

لقد رأينا أنه يمكن كتابة برنامج البسكال بسهولة على هيئة أجزاء أو إجراءات modules تسمح بتجزئة المشكلة الكلية الى سلسلة من المشاكل الجزئية . ويقدم استخدام الأجزاء ميزتين هامتين للغة البسكال . أولا بالنسبة للأنشطة التى يجب تكرارها أكثر من مرة واحدة ، نجد أن استخدام الأجزاء يلغى الحاجة إلى تكرار برمجة هذه الأنشطة بعدد المرات اللازم . وبدلا من ذلك يمكن تعريف جزء البرنامج مرة واحدة ، ثم يتم الاتصال به من أماكن عديدة مختلفة فى البرنامج . ويمكن تشغيل مجموعة بيانات مختلفة فى كل مرة يتم فيها الاتصال بهذا الجزء . وعلى هذا ... يمكن لاستخدام الأجزاء أن يقلل من طول البرنامج .

كما أن الوضوح فى المنطق الناتج من تجزئة البرنامج إلى أجزاء فردية ، حيث يمثل كل جزء شيئا محددا من المشكلة الكلية له دلالة أكبر فى واقع الأمر . ومثل هذه البرامج سهلة فى كتابتها وفى تصحيحها وفى تكوينها المنطقى عن البرامج التى تفتقد هذا النوع من التكوين . وهذا صحيح بصفة خاصة بالنسبة للبرامج الطويلة والمعقدة . وعلى هذا ... فالعديد من برامج البسكال تستخدم على ذلك الأجزاء ، بالرغم من أنها قد لاتحتوى على تكرارات لنفس الأنشطة . وفى واقع الأمر ، فإن تجزئة البرنامج إلى أجزاء فردية يعتبر بصفة عامة جزءا مهما من البرمجة العملية الجيدة .

وهناك نوعان من أنواع تجزئة البرنامج فى البسكال ، وهما الإجراءات procedures ، والدوال functions . وهذان النوعان متشابهان من تكوينهما . ويمكن الاتصال بهما على أية حال بطريقة مختلفة ، وكل منهما يتبادل المعلومات بطريقة مختلفة . وقد ناقشنا بالفعل استخدام إجراءات قياسية ( مثل read و write ) ، ودوال قياسية ( مثل sqr ) فى الفصل الثانى . دعنا نعتبر الآن الإجراءات والدوال بتفاصيل أكبر . سوف نرى بصفة خاصة كيف تكتب الإجراءات ، وماهى الاختلافات بين الإجراءات والدوال ، وكيف يمكن استخدام كل منهما استخداما مناسبا . وسوف نعتبر فى نهاية الفصل خاصية مهمة للغة البسكال ، تعرف بأنها الإعادة الذاتية recursion ، والتى تجعل من الممكن للإجراء أو للدالة أن تصل إلى نفسها بنجاح .

## ١ - الإجراءات : 1. PROCEDURES

الإجراء procedure هو تكوين برنامج فى حد ذاته ، محتوى داخل برنامج البسكال . وفى بعض لغات البرمجة الأخرى يسمى هذا المكون ببرنامج فرعى subroutine أو إجراء فرعى .

ويمكن الإشارة إلى الإجراء بكتابة اسمه ببساطة ، تتلوه قائمة اختيارية بالمؤشرات parameters . ويجب أن توضع المؤشرات بين قوسين ، وإذا ماكان هناك أكثر من مؤشر واحد ، فتفصل المؤشرات بواسطة فواصل . ودلائل الإجراء تعرف بأنها موصلات accesses للاجراء ، أو استدعاءات calls للإجراء .

وعند الإشارة إلى إجراء ينتقل التحكم تلقائيا إلى بداية الإجراء . وتنفذ العبارات الإجرائية المحتواه action statements داخل الإجراء عند ذلك ، مع الأخذ فى الاعتبار أى توضيحات خاصة تكون فردية بالنسبة للإجراء . وعند الانتهاء من تنفيذ العبارات الإجرائية الموجودة فى الإجراء يعود التحكم تلقائيا إلى العبارة التى تلى العبارة التى أشارت إلى الإجراء مباشرة .

## مــثال (٧-١)

يقرأ أحد برامج البسكال ثلاثة كميات صحيحة ، ثم يحدد أكبر كمية من هذه الكميات . ويحتوى البرنامج على إجراء اسمه maximum ، والذى يحدد أكبر كمية من الثلاث كميات ، ثم يخرجها كنتيجة للبرنامج . والعبارات الإجرائية التى تقرأ الثلاث كميات ، ثم تتصل بالإجراء موضحة أدناه .

```
BEGIN
   readln(a,b,c);
   WHILE a <> 0 DO
      BEGIN maximum; readln(a,b,c) END
END.
```

وعند ظهور إشارة إلى الإجزاء maxi mum ينتقل التحكم تلقائيا إلى هذا الإجراء ( الإجراء نفسه غير موضح فى المثال ) . ويحدد الإجراء اى من المتغيرات c , b , a هو الأكبر ، ثم يطبع النتائج . بعد ذلك يعود التحكم إلى عبارة readln التالية ( والتى تشير أيضا إلى الإجراء ) .

لاحظ أن البرنامج يقرأ مجموعات متتالية من ثلاث كميات صحيحة ، ويحدد أكبر كمية من هذه الكميات الثلاث فى المجموعة . ويستمر التكرار الذى يتحكم فيها مكون WHILE-DO حتى تتحدد القيمة 0 للمتغير a .

دعنا نعتبر الآن أن الإجراء نفسه مكتوب . وكل إجراء له عنوان خاص به وكتلة block أو مجموعة . ويكتب العنوان على النحو التالى :

```
PROCEDURE name
```

أو إذا مااستخدمت مؤشرات رسمية ، فإنه يكون على النحو التالى :

```
PROCEDURE name(formal parameters)
```

( وسوف نناقش استخدام المؤشرات الرسمية فى قسم ٧ - ٣ )

وتحتوى الكتلة على جزء توضيح ( وهو جزء محلى للإجراء ) ، ومجموعه من العبارات الإجرائية ، مثل برنامج البسكال الموجود به هذا الإجراء . وعلى هذا ... فيمكن التفكير فى الإجراء على أنه نوع خاص من أنواع برامج البسكال يكون موجودا داخل برنامج البسكال .

## مــثال (٧-٢)

فيما يلى إجراء اسمه maximum يحدد اكبر كمية من ثلاث كميات صحيحة هى c , b , a ، ويكتب النتيجة .

```
PROCEDURE maximum;

(* This procedure finds the largest
      of three integer quantities *)
VAR max : integer;
BEGIN
   IF a > b THEN max := a ELSE max := b;
   IF c > max THEN max := c;
   writeln(' The maximum is ',max)
END;
```

أول سطر يحتوى على الكلمة الأساسية PROCEDURE ، وهو عنوان للإجراء . ( لاحظ التشابه مع عنوان البرنامج ) . ويتبع هذا السطر كتلة الإجراء ، والتى تحتوى على توضيح لمتغير واحد وعدة عبارات إجرائية .

لاحظ أن هذا الإجراء يتطلب أن تكون قيم c ، b ، a محددة مسبقا قبل الاتصال بالإجراء . واستخدام الإجراء متوافق مع جزء البرنامج الموضح فى مثال ٧ – ١ .

كل توضيحات الإجراء مثل التوضيحات المذكورة فى المثال السابق يجب أن توجد داخل المجموعات المنادية calling block ( أو داخل مجموعة خارجية تحيط بالمجموعة المنادية ) . ويجب أن تكون توضيحات الاجراء والدالة بصفة خاصة آخر عناصر فى قسم التوضيحات ، وتتبع توضيحات المتغيرات ( انظر قسم ٥ – ١ ، وقسم ٢ – ٥ ) . وبعد توضيح الإجراء داخل المجموعة ، يمكن الاتصال به من أى مكان فى المجموعة .

مثال (٧-٣)

*أكبر عدد من ثلاثة أعداد* . فيما يلى برنامجاً كاملاً بلغة البسكال ، يدمج أجزاء البرنامج التى سبق ذكرها فى المثالين السابقين

```
PROGRAM sample(input,output);
(* This program uses a procedure to determine the maximum
             in each set of three integer quantities *)
VAR a,b,c : integer;

   PROCEDURE maximum;
   (* This procedure finds the largest
          of three integer quantities *)
   VAR max : integer;
   BEGIN
      IF a > b THEN max := a ELSE max := b;
      IF c > max THEN max := c;
      writeln(' The maximum is ',max)
   END;

BEGIN   (* main action block *)
   readln(a,b,c);
   WHILE a <> 0 DO
      BEGIN maximum; readln(a,b,c) END
END.
```

يكرر هذا البرنامج قراءة مجموعات من ثلاثة أعداد صحيحة ، ويحدد أكبر عدد من هذه الاعداد الثلاثة فى كل مجموعة . ويستمر الكمبيوتر فى قراءة وتشغيل مجموعات متتالية من البيانات ، طالما أنه لم يتعرض لقراءة صفر ( حيث إن الصفر يعنى انتهاء هذه المجموعات ) .

لاحظ أن العملية قد تم توضيحها قبل الاتصال بها ، كما أن توضيح الإجراء تلى توضيحات المتغيرات . ( الأسطر الفارغة ظهرت لتحسين قراءة البرنامج فقط ) . ولاحظ أيضا أن المتغيرات c , b , a التى سبق تعريفها فى الجزء الرئيسى للبرنامج ، تم تمييزها خلال البرنامج ، بما فيها الإجراء . ومن ناحية أخرى ... تم تعريف max داخل الإجراء فقط . وعلى هذا ... فليس من الممكن استخدام المتغير max داخل الجزء الرئيسى للبرنامج .

والقواعد التى تحكم أدلة الإشارة للإجراء ، تكون عامة فى الواقع عما تحدد الأمثلة سالفة الذكر . وسوف نرى بإيجاز أنه يمكن الاتصال بالإجراء عن طريق الجزء الرئيسى من البرنامج أكثر من مرة واحدة . وأكثر من هذا ... يمكن أن تتصل إجراءات أخرى ودوال أخرى أيضا بالإجراء .

## ٧ – مدى المعرفات : 7. SCOPE OF IDENTIFIERS

يمكن أن توضح الثوابت والمتغيرات التى تظهر داخل العبارات الإجرائية لأداء إجراء معين خارجيا ويكون ذلك داخل مجموعة من مجموعات البرنامج ، والتى تحتوى على توضيح الإجراء نفسه . أو يمكن أن يحدث ذلك محليا ، أى داخل الإجراء نفسه . ويمكن استخدام هذه الثوابت والمتغيرات التى توضح فى المجموعة المحتوية على توضيح الإجراء فى أى مكان داخل هذه المجموعة ، سواء أكان ذلك داخل الإجراء ، أم خارجه . وتعتبر المعرفات التى تعرف بهذه الطريقة أنها شاملة global للإجراء ومن ناحية أخرى ... فإن الثوابت والمتغيرات المحلية local لاتعرف خارج الإجراء . وعلى هذا ... فلايمكن استخدامها خارجه .

ومدى scope المعرف يشير إلى المنطقة التى يوضح داخلها المعرف . وعلى هذا ... يمكن استخدامه داخلها . ويطبق هذا المفهوم على كل أنواع التوضيحات ، وليس على الثوابت والمتغيرات وحدها .

### مثال (٧-٤)

*أكبر عدد من ثلاثة أعداد* . دعنا نفحص مرة أخرى برنامج البسكال الموجود فى مثال ( ٧ – ٣ )

```
PROGRAM sample(input,output);
(* This program uses a procedure to determine the maximum
                in each set of three integer quantities *)

VAR a,b,c : integer;

   PROCEDURE maximum;
   (* This procedure finds the maximum
         of three integer quantities *)
   VAR max : integer;
   BEGIN
      IF a > b THEN max := a ELSE max := b;
      IF c > max THEN max := c;
      writeln(' The maximum is ',max)
   END;

BEGIN   (* main action block *)
   readln(a,b,c);
   WHILE a <> 0 DO
      BEGIN maximum; readln(a,b,c) END
END.
```

يحتوى هذا البرنامج على إجراء واحد يسمى maximum . وتوضح المتغيرات c , b , a خارج هذا الإجراء . وعلى هذا ... فهى شاملة بالنسبة لهذا الإجراء . ويمكن استخدام هذه المتغيرات داخل الإجراء وخارجه ( كما هو الحال فى هذا المثال ) . وعلى أية حال ... فإن المتغير max تم توضيحه داخل الإجراء . وعلى هذا ... فإن max متغير محلى لهذا الإجراء ، ولايمكن استخدامه خارجه .

وبصفة عامة ... تفضل المعرفات المحلية عن المعرفات الشاملة . وهذا لايتعارض مع المنطق الشامل للبرنامج . ويسهم استخدام المعرفات المحلية بصفة خاصة فى جعل البرنامج أكثر وضوحا ، كما أنه يقلل أيضا من فرصة حدوث أخطاء برمجة ، بسبب أدلة الإشارة غير الصحيحة أو غير المتناسقة .

ومن ناحية أخرى ... يتطلب الكثير من البرامج أن تميز عناصر بيانات محددة داخل وخارج الإجراء . وإحدى طرق نقل مثل هذه المعلومات عبر حدود البرنامج هى استخدام المعرفات الشاملة ، حيث يمكن الإشارة إلى المعرفات الشاملة من أى مكان كلما دعت الحاجة لذلك . ( وهناك طريقة أخرى ، وهى استخدام مؤشرات parameters كما يتضح ذلك من القسم التالى ) . وعلى هذا ... يمكن أن يحتوى البرنامج على كل من المعرفات الشاملة والمحلية ، طبقا للمتطلبات المنطقية للمشكلة وللتكوين المناظر للبرنامج . ومن المهم تمييز وقت استخدام كل نوع من أنواع المعرفات

ومن الممكن استخدام نفس المعرف لتمثيل كينونات مختلفة فى أجزاء مختلفة من أجزاء البرنامج ( بالرغم من أن هذا يعتبر برمجة ضعيفة عمليا ) . وعلى هذا ... فالمعرف الذى يتم توضيحه بأنه محلى داخل الإجراء ، يمكن أن يكون له نفس الاسم كمعرف شامل يوضح خارجيا . وفى هذه الحالات فإن التعريف المحلى يكون له الأولوية داخل مدى المعرف . وخارج هذا المدى لايكون المعرف المحلي معرفا ، ولايطبق إلا التعريف الشامل .

## مـثال (٧-٥)

والشكل الهيكلى التالى لبرنامج بسكال يحتوى على إجرائين :

```
PROGRAM sample(input,output);
VAR a,b : integer;
    c,d : char;

     PROCEDURE one;
     VAR a,d : real;
     BEGIN
         .
         .
         .
     END;

     PROCEDURE two;
     VAR a : char;
         b : boolean;
     BEGIN
         .
         .
         .
     END;

BEGIN   (* main program - action statements *)
   .
   .
   .
END.
```

لاحظ أن الإجراء الأول يحتوى على متغيرين حقيقيين محليين ، هما : d , a . ويمكن لهذا الإجراء أن يشير أيضا إلى متغيرين شاملين ، هما : b وهو صحيح ، و c وهو حرفى .

ويميز الإجراء الثانى المتغيرين المحليين b , a ، وهما من النوع الحرفى والبوليان على التوالى . كما يمكن للإجراء الثانى أن يشير أيضا إلى المتغيرين d, c وهما حرفيان .

وتحتوى المجموعة الأساسية للبرنامج على أربعة متغيرات محلية b , a ، وهما صحيحان ، و d , c وهما حرفيان .

لاحظ أن العديد من أسماء المتغيرات يستخدم بطريقة مختلفة داخل الإجراءات ، عنها فى الجزء الرئيسى للبرنامج . وبصفة خاصة تم إعادة تعريف a داخل كل إجراء ، وتم إعادة تعريف كل من d , b داخل إجراء واحد من الإجرائين . وتأخذ التعريفات المحلية الأولوية داخل الإجراءات المناظرة لها . أما داخل الجزء الرئيسى من البرنامج ، فسوف تفسر المتغيرات طبقا لتوضيحها الأصلى .

ويطبق موضوع التوضيحات المحلية والشاملة على الإجراءات ( والدوال ) وعلى الثوابت والمتغيرات . وعلى هذا ... فمن الممكن توضيح إجراء ( أو دالة ) داخل إجراء آخر . وتسمح هذه السمة بتداخل الإجراءات داخل بعضها . وفى مثل هذه الحالات يكون من المهم بالطبع ألا يحدث اتصال بالإجراء خارج المجموعة التى تحتوى على توضيح الإجراء نفسه .

## مثال (٧-٦)

فيما يلى شكلا هيكليًا لبرنامج بسكال يحتوى على إجراءات متداخلة . ومدى كل إجراء مختلف موضح عن طريق الترحيل .

```
PROGRAM main(input,output);

   PROCEDURE one;

      PROCEDURE two;
      BEGIN    (* procedure two - action statements *)
         .
         .
         .
      END;

      PROCEDURE three;
      BEGIN    (* procedure three - action statements *)
         .
         .
         two;   (* reference to procedure two *)
         .
         .
      END;

      BEGIN    (* procedure one - action statements *)
         .
         .
         two;   (* reference to procedure two *)
         three;   (* reference to procedure three *)
         .
         .
      END;

   BEGIN    (* main program - action statements *)
      .
      .
      one;   (* reference to procedure one *)
      .
      .
   END.
```

فى هذا المثال نجد أن الإجراء الأول موضح داخل الجزء الرئيسى للبرنامج . أما الإجراءان two , three ، فموضحان داخل الإجراء one . وعلى هذا فإن مدى الإجرائين two , three يكون محليا للإجراء one . وعلى هذا ... فيمكن الاتصال بالإجرائين two , three داخل الإجراء one ، وليس داخل العبارات الاجرائية للجزء الرئيسى للبرنامج . كما يمكن للإجرائين two , three أن يتصلا ببعضهما أيضا . ومن ناحية أخرى ... يمكن الاتصال بالإجراء one بواسطة العبارات الإجرائية للجزء الرئيسى للبرنامج ، حيث إنها موضحة داخل هذا الجزء .

مـثال ( ٧ - ٧ )

*أكبر عدد من ثلاثة أعداد* . نعتبر الآن صيغة مختلفة لبرنامج البسكال الموجود فى مثال ( ٧ - ٤ ) . وفى هذا المثال نجزئ الإجراء الأصلى maximum إلى إجرائين ، أحدهما داخل الآخر ( كما هو موضح بواسطة الترحيل ) . ويسمى الإجراء الخارجى بنفس الاسم maximum ، إلا أن بعض الجمل الإجرائية ظهرت الآن داخل الإجراء الداخلى findmax .

```
PROGRAM sample(input,output);

(* This program uses a procedure to determine the maximum
            in each set of three integer quantities *)

VAR a,b,c : integer;

    PROCEDURE maximum;
    (* This procedure finds the largest
          of three integer quantities *)
    VAR max : integer;

        PROCEDURE findmax;
        (* This is where the action is *)
        BEGIN
           IF a > b THEN max := a ELSE max := b;
           IF c > max THEN max := c
        END;

    BEGIN   (* back to maximum - action statements *)
       findmax;
       writeln(' The maximum is ',max)
    END;

BEGIN   (* main action statements *)
   readln(a,b,c);
   WHILE a <> 0 DO
      BEGIN maximum; readln(a,b,c) END
END.
```

لاحظ أن findmax تم توضيحه داخل maximum ، كما أن maximum تم توضيحه داخل الجزء الرئيسى للبرنامج . وعلى هذا ... فإن findmax يكون محليا لإجراء maximum . لاحظ أيضا أنه يتم الاتصال بإجراء maximum من الجزء الرئيسى للبرنامج ، بينما يتم الاتصال بإجراء findmax من داخل الإجراء maximum .

وتظهر حالة شبيهة بالنسبة للمتغيرات a - b - c , max . فالمتغيرات a - b - c بصفة خاصة هى شاملة بالنسبة للإجرائين ، حيث إنه تم توضيحها داخل الجزء الرئيسى للبرنامج . ويمكن استخدام هذه المتغيرات الثلاثة على أية حال فى أى مكان فى البرنامج . ومن ناحية أخرى ... فإن المتغير max يعد متغيراً محليًا للإجراء maximum ويمكن على ذلك استخدامه داخل الإجراء maximum ، أو داخل الإجراء الداخلى findmax فقط .

وأخيرا يجب ملاحظة مفهوم المدى مع عناوين العبارات ومع الثوابت والمتغيرات والإجراءات ( والدوال ) . ولايمكن وضع عنوان بصفة خاصة لعبارة إلا إذا وجدت العبارة داخل مدى توضيح العنوان . كما أنه لايمكن نقل التحكم إلى عبارة لها عنوان من خارج مدى هذا العنوان . وأنه من الممكن على أية حال نقل التحكم إلى عبارة لها عنوان من أى مكان آخر داخل مدى العنوان . وعلى هذا ... يمكن نقل التحكم خارج الإجراء ( أو الدالة ) إذا ماحدث النقل خارج مدى تعريف العنوان . وهذا يطبق أيضا على الإجراءات المتداخلة لأى عدد من مرات التداخل .

## مـــثال (٧-٨)

*حساب متوسط قائمة من الأعداد* . فى مثال ٦ -٢٦رأينا برنامج بسكال يحسب متوسط قائمة بالأعداد ، وذلك باستخدام عبارة GOTO لنقل التحكم خارج دورة الشرط عندما يقرأ الكمبيوتر عددا سالبا . ( العدد السالب يعبر عن تحقق شرط التوقف ، ولايحسب فى حساب المتوسط ) .

ولنعد كتابة هذا البرنامج الآن ، بحيث إن بيانات المدخلات ودوال حساب البيانات تعامل عن طريق إجراء . وسوف نظل محتفظين باستخدام عبارة GOTO ، بحيث ينقل التحكم خارج الإجراء عند إدخال عدد سالب داخل الكمبيوتر . وفيما يلى برنامجا كاملا بلغة البسكال لأداء ذلك .

```
PROGRAM average6(input,output);

(* THIS PROGRAM CALCULATES THE AVERAGE OF A LIST OF NUMBERS
   USING THE WHILE - DO STRUCTURE AND A GOTO STATEMENT
   WITHIN A PROCEDURE *)

LABEL 10;
VAR count : integer;
    sum,average : real;

    PROCEDURE enterdata;
    (* This procedure reads and sums
       successive real, positive quantities *)
    VAR x : real;
        flag : boolean;
    BEGIN   (* action statements *)
       flag := true;
       WHILE flag DO
          BEGIN
             read(x);
             IF x < 0 THEN GOTO 10;
             sum := sum+x;
             count := count+1
          END
     END;

BEGIN   (* main action statements *)
     count := 0;
     sum := 0;
     enterdata;
10 : average := sum/count;
     writeln(' The average of ',count:5,' numbers is ',average)
END.
```

لاحظ أن العبارة رقم 10 والمتغيرات average , sum , count تم توضيحها داخل الجزء الرئيسى للبرنامج ، حيث إن هذه العناصر مطلوبة فى كل من جزء الأداء لهذا الجزء ، وفى الإجراء أيضا . ومن ناحية أخرى ... فإن flag و x تم توضيحهما داخل الإجراء محليا ، وذلك نظرا لأن هذه المعرفات غير مطلوبة خارج الإجراء . ولاحظ أن التحكم ينقل خارج الإجراء مباشرة عند ظهور قيمة سالبة للمعرف x .

ويجب أن يكون مفهوما أن هذا المثال يحتوى على العديد من الأشياء المريحة لتوضيح سمات معينة . وفى الواقع العملى يجب أن يكتب هذا البرنامج دون استخدام عبارة GOTO . والأكثر من هذا ... يجب أن تظهر حسابات المتوسط وعبارات المخرجات التالية لها داخل الإجراء . وهذا يسمح بتوضيح كل المتغيرات داخل الإجراء محليا . وهو اتجاه أفضل لحل المشكلة .

## ٣ - المؤشرات : 3. PARAMETERS

يتطلب الكثير من برامج البسكال أن يحدث تبادل للمعلومات بين إجراء ( أو دالة ) وبين النقطة التى حدث عندها إشارة للإجراء ( أو للدالة ) ، وإحدى طرق تحقيق ذلك هى استخدام المتغيرات الشاملة كما سبق ذكره فى القسم السابق . كما سبق أن رأينا على أية حال أنه هناك وجهات نظر لاتحبب من استخدام المتغيرات الشاملة . فمثلا يمكن لتغيير قيمة متغير شامل داخل أحد الإجراءات أن ينتج عنها تغيير معلومات أخرى خارج الإجراء سهوا ، والعكس صحيح أيضا . وأكثر من هذا ... فإن نقل مجموعات بيانات متعددة لايمكن تحقيقه بسهولة باستخدام المتغيرات الشاملة .

ويقدم استخدام المؤشرات طريقة أفضل لتبادل المعلومات بين إجراء والنقطة المشيرة إليه ، فينقل كل عنصر بيانات بين المؤشر الفعلى actual parmeter والموجود داخل دليل ( أو المشير إلى ) الإجراء والمؤشر الرسمى formal parmeter المناظر والمعرف داخل الإجراء نفسه . وعند الاتصال بالإجراء تحل المؤشرات الفعلية محل المؤشرات الرسمية منتجة آلية تبادل المعلومات بين الإجراء ونقطة الإشارة إليه . وتعتمد هذه الطريقة التى يحدث بها النقل على طريقة تعريف واستخدام المؤشرات .

مـثال (٧-٩)

فيما يلى تخطيطا هيكليا يوضح أبسط أنواع تبادل المعلومات بين الإجراء وبين دليله :

```
PROGRAM sample(input,output);
VAR a,b,c,d : real;

    PROCEDURE flash(x,y : real);
    BEGIN
        .
        .
    (* process the values of x and y *)
        .
        .
    END;

BEGIN   (* main program action statements *)
   .
   .
   flash(a,b);
   .
   .
   flash(c,d);
   .
   .
END.
```

فى هذا المثال المتغيرات الحقيقية y , x هى مؤشرات رسمية معرفة داخل الإجراء flash . أما المؤشرات الفعلية ، فهى المتغيرات الحقيقية d , c , b , a . نفترض أنه تم تحديد قيم فى مكان مافى البرنامج لكل من d , c , b , a وذلك قبل الإشارة إلى الإجراء .

تتسبب أول إشارة إلى الإجراء فى نقل قيم المؤشرات الفعلية b , a إلى المتغيرات الرسمية y , x . وعلى هذا ... فإن قيم b , a تمر عبر الإجراء flash ، حيث يتم تشغيلها هناك . ( وطريقة تشغيلها غير موضحة فى هذا المثال ) .

وتعاد هذه العملية فى عبارة الإجراء الثانية ، مع نقل قيم d , c فى هذه المرة إلى y , x . وعلى هذا ... فتنقل قيم d , c عبر الإجراء flash ، حيث يتم تشغيلها أيضا .

لاحظ أننا قمنا بتشغيل مجموعتين مختلفتين من البيانات ببساطة ، وذلك عن طريق الاتصال بنفس الإجراء مرتين ، وذلك بمجموعتين مختلفتين من المؤشرات الفعلية فى كل مرة .

وهناك قواعد معينة يجب أن تراعى عند إعداد مناظرة بين دليل ( أو الإشارة إلى ) الإجراء ، وبين الإجراء نفسه . ( أى عند التعويض بالمؤشرات الفعلية فى المؤشرات الرسمية ) ، وهى مايلى :

(١) يجب أن يكون عدد المؤشرات الفعلية فى دليل الإجراء ، هو نفس عدد المؤشرات الرسمية الموجودة فى تعريف الإجراء .

(٢) يجب أن يكون لكل مؤشر فعلى نفس النوع ، مثل المؤشر الرسمى المناظر له .

(٣) يجب التعبير عن كل مؤشر فعلى بطريقة متناسقة مع المؤشر الرسمى المناظر له كما تحدده فئة المؤشر الرسمى ( وفئة المؤشر الرسمى موصوفة أدناه ) .

### مثال (٧-١٠)

اعتبر التكوين الهيكلى للبرنامج التالى :

```
PROGRAM sample(input,output);
VAR a,b : integer;
    c,d : real;

    PROCEDURE flash(x : integer; y : real);
    BEGIN
      .
      .
    (* process the values of x and y *)
      .
      .
    END;

BEGIN   (* main program action statements *)
  .
  .
  flash(a,c);
  .
  .
  flash(b,d);
  .
  .
END.
```

لاحظ أن كل دليل إجراء يحتوى على مؤشرين فعليين ، حيث تم تعريف مؤشرين رسميين ( y , x ) داخل الإجراء . وأكثر من هذا ... فإن x موضح بأنه متغير صحيح ، و y موضح بأنه متغير حقيقى . وعلى هذا ... فكل دليل لإجراء يجب أن يحتوى على متغير صحيح ومتغير حقيقى ، وبنفس هذا الترتيب .

ويمكن أن يحتوى الإجراء على أربع فئات من المؤشرات الرسمية ، وهى مؤشرات قيمة value parameters ومؤشرات متغير varible parameters ومؤشرات إجراء procedure parameters ومؤشرات دالة function parameters . وسوف نناقش مؤشرات القيمة ومؤشرات المتغير الآن . كما أن مؤشرات الإجراء ومؤشرات الدالة سوف تناقش فيما بعد فى هذا الفصل ( انظر القسم ٧ – ٥ ) .

## مؤشرات القيمة : Value Parameters

يمكن التفكير فى مؤشرات القيمة بصورة أفضل على أنها مؤشرات مدخلات input parameters ، وذلك بالنسبة إلى إجراءاتها المناظرة لها . ويشمل استخدام مؤشرات القيمة نقل قيمة بدلا من التعويض بالمؤشر الفعلى . وعلى هذا ... فعند نقل المعلومات بين مؤشر فعلى ومؤشر قيمة ، فإن قيمة المؤشر الفعلى تتحدد لمؤشر القيمة . وعند ذلك يمكن تشغيل هذه القيمة داخل الإجراء ( بالإشارة إلى مؤشر القيمة ) والقيم الممثلة بواسطة مؤشرات قيمة لايمكن على أية حال نقلها فى الاتجاه العكسى ، أى من الإجراء إلى الجزء المنادى على الإجراء من البرنامج . وهذا هو سبب الإشارة إلى مؤشرات القيمة بأنها مؤشرات مدخلات .

ومؤشرات القيمة سهلة جدا فى استخدامها ... فيتم توضيحها بذكر أسمائها ، وأنواع البيانات المناظرة لها ببساطة داخل عنوان الإجراء ، وذلك دون ذكر أى سابقة prefix ( مثل VAR ) . وغياب مثل هذه السابقة هو الذى يعرف هذه الفئة من المؤشرات ببساطة . وتستخدم مؤشرات القيمة فى كل من مثالى ٧ – ٩ و ٧ –١٠ .

ويجب أن يكون مفهوما أن أى تبديل فى قيمة مؤشر القيمة داخل الإجراء لن يؤثر على قيمة أى مؤشر من المؤشرات الفعلية . ( تذكر أن المؤشرات الفعلية يشار إليها بأنها مؤشرات مدخلات input ) . وهذه الخاصية قد تحد من استخدام مؤشرات القيمة . وعلى أية حال ... فمثل هذه المؤشرات سهلة فى استخدامها فى الحالات التى يسمح فيها بتبادل المعلومات فى اتجاه واحد . وأكثر من هذا ... فحيث إن قيم values المؤشرات الفعلية هى التى تنقل بدلا من المؤشرات نفسها ، فهناك عمق ممكن فى طريقة كتابة المؤشرات الفعلية . ويمكن التعبير عن المؤشرات الفعلية بصفة خاصة كثوابت أو متغيرات أو تعبيرات ( مع الحفاظ على أن قيمة المؤشر تكون من النوع المناسب ) .

### مثال (٧-١١)

اعتبر التعديل التالى فى البرنامج الذى سبق توضيح تخطيطه فى مثال ٧ –١٠

```
PROGRAM sample(input,output);
VAR a,b : integer;
    c,d : real;

     PROCEDURE flash(x : integer; y : real);
     BEGIN
        .
        .
     (* process the values of x and y *)
        .
        .
     END;
```

```
BEGIN   (* main program action statements *)
   .
   .
   flash(3,a*(c+d)/b);
   .
   .
   flash(2*(a+b),-0.5);
   .
   .
END.
```

لاحظ أن المؤشرات الرسمية y , x تم توضيحها فى الإجراء flash ، وأنها مؤشرات قيمة . ( وهذا صحيح أيضا فى مثالى ٧ – ٩ و ٧ –١٠) . وأولها ( x ) هو من النوع الصحيح . أما ثانيهما ( y ) فهو من النوع الحقيقى . وعلى هذا ... فكل إشارة إلى flash يجب أن تحتوى على مؤشرين فعليين ، أولهما يجب أن يكون صحيحا ، وثانيهما يجب أن يكون حقيقيا

ويحتوى الجزء الرئيسى للبرنامج على إشارتين مختلفتين للإجراء ( أى إشارتين مختلفتين لـ flash ) . وتحتوى كل إشارة للإجراء على مؤشرين فعليين ، أولهما من النوع الصحيح ، وثانيهما من النوع الحقيقى كما هو مطلوب . لاحظ أن اثنين من المؤشرات كتبا كثابتين ، واثنين كتبا كتعبيرين . وعلى هذا ... ينقل أول دليل ( أو إشارة إلى ) للإجراء القيمة 3 إلى x والقيمة الخاصة بالتعبير الحقيقى a*( c+d ) / b إلى y . وبالمثل ينقل الدليل الثانى للإجراء قيمة التعبير الصحيح 2* (a+b) إلى x وقيمة 0.5- إلى y .

## مثال (٧-١٢)

*حساب الاستهلاك* . مرة أخرى اعتبر المشكلة الموصوفة فى مثال ٦ –١٧لحساب الاستهلاك بإحدى الطرق الثلاث لحساب الاستهلاك . دعنا نعيد الآن كتابة البرنامج مع استخدام إجراء منفصل لكل طريقة من هذه الطرق . يقدم هذا الأسلوب طريقة أوضح لتنظيم البرنامج طبقا لأجزائه المنطقية . ( وفى المثال ٦ –١٧يوجد وصف تفصيلى لمنطق البرنامج ) . وفيما يلى برنامج البسكال كاملا لأداء هذه العمليات .

```
PROGRAM depreciation2(input,output);

(* THIS PROGRAM USES PROCEDURES TO CALCULATE
   DEPRECIATION BY ONE OF THREE POSSIBLE METHODS *)

VAR years,choice : integer;
    value : real;
    flag : boolean;

    PROCEDURE straightline(n : integer; val : real);
    (* Calculate depreciation using the straight line method *)
    VAR year : integer;
        deprec : real;
    BEGIN
       writeln(' Straight-Line Method');
       writeln;
       deprec := val/n;
       FOR year := 1 TO n DO
          BEGIN
             val := val-deprec;
             write(' End of Year ',year:2);
             write('  Depreciation: ',deprec:5:0);
             writeln('  Current Value: ',val:6:0)
          END
    END;
```

(تكملة البرنامج فى الصفحة التالية)

```
PROCEDURE decliningbalance(n : integer; val : real);
(* Calculate depreciation using the
    double-declining balance method *)
VAR year : integer;
    deprec : real;
BEGIN
   writeln(' Double Declining Balance Method');
   writeln;
   FOR year := 1 TO n DO
      BEGIN
         deprec := 2*val/n;
         val := val-deprec;
         write(' End of Year ',year:2);
         write('   Depreciation: ',deprec:5:0);
         writeln('   Current Value: ',val:6:0)
      END
END;

PROCEDURE sumofyears(n : integer; val : real);
(* Calculate depreciation using the
   sum-of-the-years'-digits method *)
VAR year : integer;
    deprec,tag : real;
BEGIN
   writeln(' Sum-of-the-Years''-Digits Method');
   writeln;
   tag := val;
   FOR year := 1 TO n DO
      BEGIN
         deprec := (n-year+1)*tag/(n*(n+1)/2);
         val := val-deprec;
         write(' End of Year ',year:2);
         write('   Depreciation: ',deprec:5:0);
         writeln('   Current Value: ',val:6:0)
      END
END;

BEGIN   (* main action block *)
   flag := true;
   REPEAT
      page;      (* begin input routine *)
      write(' Method: (1-SL  2-DDB  3-SYD  4-End) ');
      readln(choice);
      IF choice <> 4 THEN
      BEGIN
         write(' Original value: ');
         readln(value);
         write(' Number of years: ');
         readln(years);
         writeln
      END;   (* input routine *)
      CASE choice OF
         1 : straightline(years,value);
         2 : decliningbalance(years,value);
```

(تكملة البرنامج فى الصفحة التالية)

```
                3 : sumofyears(years,value);
                4 : BEGIN
                      writeln(' That''s all, folks!');
                      flag := false
                    END
              END   (* choice *)
           UNTIL flag = false
         END.
```

لاحظ أن مكون CASE مازال مستخدما كما فى مثال ٦ -١٧إلا أن البرنامج يستخدم الآن إجراء مختلفا لكل نوع من أنواع الحسابات . ويجب أن تستخدم الإجراءات مؤشرات القيمة val , n التى تمثل العمر الافتراضى للعنصر المستهلك وقيمته الأصلية على التوالى . والمؤشرات الفعلية المناظرة فى الجزء الرئيسى للبرنامج هى value , years . وعلى هذا ... فعند الإشارة إلى الإجراء تتحدد قيمة year للمؤشر n ، وتحدد قيمة value للمؤشر val . ( ويمكننا بالطبع استخدام نفس أسماء المتغيرات فى كل من الجزء الرئيسى للبرنامج وكل إجراء من الإجراءات . وهذا يكون أبسط ، بالرغم من أنه لايكون معطيا للمعلومات بطريقة أفضل ) .

لاحظ أن استخدام مؤشرات القيمة مناسب فى هذا المثال ، نظرا لأن الإجراءات لاتغير قيم المؤشرات ، ولاتعود بأى قيمة الى الجزء الرئيسى ( المنادى ) للبرنامج .

## مؤشرات المتغير : Variable Parameters

لقد رأينا أن مؤشرات القيمة مريحة فى العمل فى مواقف تحتاج لنقل معلومات من دليل الإجراء إلى الإجراء فقط . وفى العديد من التطبيقات – على أية حال – يجب نقل المعلومات فى كلا الاتجاهين بين الاجراء ودليله . وفى كلمات أخرى ... يجب أن يكون الإجراء قادرا على الحصول على مدخلات من الجزء المنادى عليه ، وكذلك إعادة مخرجات إلى هذا الجزء . وتستخدم مؤشرات المتغير فى هذه الحالات بصفة خاصة .

وعندما يتم الاتصال بإجراء يحتوى على مؤشرات متغير ، يتم التعويض بالمؤشر الفعلى الموجود فى الإشارة للإجراء فى المتغير الرسمى الموجود داخل الإجراء نفسه . وعلى هذا ... يستخدم المؤشر الفعلى أثناء تنفيذ الإجراء . وهذا بعكس استخدام مؤشر القيمة ، حيث إن قيمة value المؤشر الفعلى هى التى تحدد للمؤشر الرسمى ( لاحظ الفرق بين التحديد assignment و التعويض substitution ) . وعملية التعويض هذه هى التى تسمح بنقل المعلومات فى الاتجاهين بين دليل الإجراء والإجراء نفسه .

ومن ناحية أخرى ... تجب الإشارة إلى أنه يمكن تعويض متغير بمتغير آخر فقط . وعلى هذا ... فالمؤشرات الفعلية التى تعوض كمؤشرات متغير يجب أن تكون متغيرات أيضا ، ولايمكن أن تكون ثوابت أو تعبيرات . وعلى هذا ... تكون لدينا درجة عمومية أقل فى استخدام مؤشرات المتغير عن استخدام مؤشرات القيمة .

وهناك نتيجة أخرى لعملية التعويض ، وهى أن أى تغيير يحدث فى قيمة مؤشر المتغير داخل الإجراء سوف يغير من قيمة المؤشر الفعلى المناظر الموجود خارج الإجراء . وعلى هذا ... فيمكن أن تؤثر مؤشرات المتغير على البرنامج ككل ، بالرغم من أن مداها يكون محليا داخل الإجراء التى تم توضيحها فيه .

وتوضع مؤشرات المتغير داخل عنوان الإجراء كما يحدث بالنسبة لمؤشرات القيمة . وعلى أية حال ... فيجب أن يسبق توضيحات مؤشرات المتغير الكلمة الأساسية VAR كما يتضح ذلك من المثال التالى .

## مــثال (٧-١٣)

اعتبر مرة أخرى التكوين الهيكلى للبرنامج الموجود فى مثال ٧ -١٠ إذا ماعدل الإجراء ليستخدم مؤشرات متغير ، فسوف يظهر التكوين الهيكلى على النحو التالى :

```
PROGRAM sample(input,output);
VAR a,b : integer;
    c,d : real;

     PROCEDURE flash( VAR x : integer; VAR y : real);
     BEGIN
        .
        .
     (* process the values of x and y *)
        .
        .
     END;

BEGIN   (* main program action statements *)
   .
   .
   flash(a,c);
   .
   .
   flash(b,d);
   .
   .
END.
```

تتسبب أول إشارة للإجراء فى تعريض المؤشرات الفعلية c , a مكان y , x . فإذا ماتغيرت قيمة x أو y داخل الإجراء ، فإن التغيير المناظر فى قيمة c , a يحدث فى الجزء الرئيسى من البرنامج .

وبالمثل ينتج عن ثانى إشارة للإجراء تعويض d , b مكان y , x . وأى تغيير فى قيمة x أو y داخل الإجراء يتسبب فى تغيير قيمة b أو d فى الجزء الرئيسى للبرنامج .

لاحظ أن المؤشرات الفعلية تتفق مع المؤشرات الرسمية المناظرة فى عددها وفى نوعها .

## مــثال (٧-١٤)

*البحث عن أكبر قيمة* . افرض أننا نرغب فى إيجاد قيمة خاصة من قيم x ، والتى تتسبب فى تعظيم الدالة .

$$y = x \cos (x)$$

داخل منطقة محاطة بقيمة x = 0 من ناحية اليسار ، وقيمة x = π من ناحية اليمين . ونطلب أن تكون قيمة x التى تعظم الدالة معروفة بدقة . كما نطلب أيضا أن يكون مخطط البحث كفؤ نسبيا ، بمعنى أنه يجب أن يتم تقويم الدالة y = x cos (x) أقل عدد ممكن من المرات .

وكطريقة واضحة لحل هذه المشكلة ، يجب إنتاج عدد كبير من دوال متقاربة مع بعضها فى محاولات البحث ( أى تقويم الدالة عند نقاط x = 0 , x = 0.0001 , x = 0.0002 , ... , x = 3.1415 , x = 3.1416 ) وتحديد أكبر هذه

القيم بالفحص البصرى . وهذه الطريقة منخفضة الكفاءة على أية حال ، وتتطلب تدخلا آدميا للحصول على النتيجة النهائية . بدلا من ذلك ... دعنا نستخدم مخطط الحذف elimination scheme التالى ، وهو إجراء حسابات مرتفع الكفاءة لكل الدوال التى لها نهاية عظمى واحدة داخل منطقة البحث .

وسوف تجرى الحسابات كمايلى . نبدأ بنقطتى بحث عند مركز خطوة البحث ، يقعان على مسافة قريبة جدا من بعضهما كما هو موضوح فى شكل ٧ – ١ . وقد استخدمت الاصطلاحات التالية :

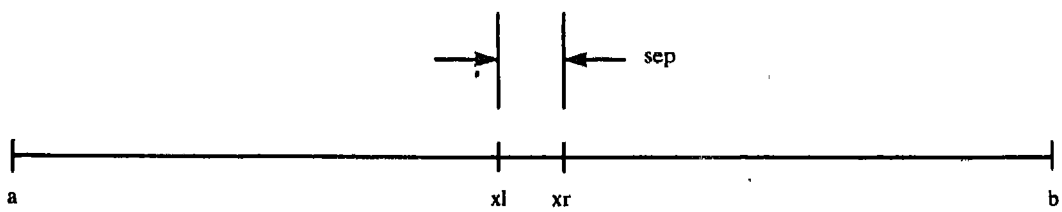
a = النهاية اليسرى لمنطقة البحث .

xl = النقطة الداخلية اليسرى للبحث .

xr = النقطة الداخلية اليمنى للبحث .

b = النهاية اليمنى لمنطقة البحث .

sep = المسافة بين xr , xl .



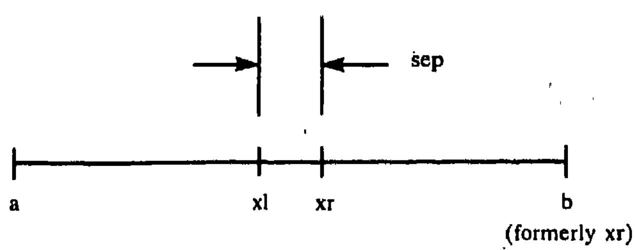

شكل ٧ – ١

فاذا ماعرفت sep , b , a ، فيمكن حساب النقاط الداخلية على النحو التالى :

```
xl = a + .5*(b-a-sep)

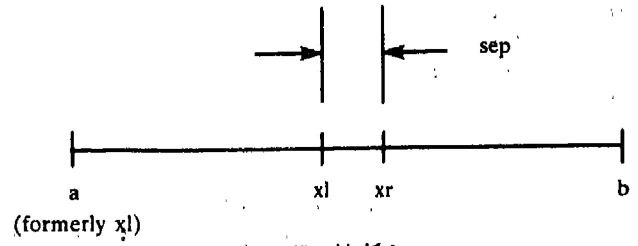
xr = a + .5*(b-a+sep) = xl + sep
```

دعنا نقوم الدالة $y = x \cos(x)$ عند xr , xl ، ونسمى هذه القيم yr , yl على التوالى . افرض أن yl كانت أكبر من yr . هذا يعنى أن أكبر قيمة تقع بين xr , a . وعلى هذا ... فإننا نحتفظ بجزء المنطقة الذى يتراوح من x = a إلى x = xr للاستمرار فى البحث ( وسوف نشير إلى النقطة xr القديمة حاليا بأنها b ، حيث إنها هى نقطة النهاية اليمنى لمنطقة البحث ) وننتج نقطتين جديدتين للبحث ، وهما : xr , xl . وتقع هاتان النقطتان فى مركز منطقة البحث الجديدة بعيدتين عن بعضهما مسافة sep كما هو موضح فى شكل ٧ – ٢ .




شكل ٧ – ٢

ومن ناحية أخرى ... افرض أننا فى منطقة بحثنا الأصلى original كانت yr أكبر من yl . سوف يحدد ذلك أن منطقة البحث الجديدة يجب أن تقع بين b , xl . وعلى هذا ... فإننا نعيد تسمية النقطة التى كانت تسمى xl فى الأصل بأنها النقطة x ، ثم ننتج نقطتى بحث xr , xl عند مركز منطقة البحث الجديدة ، كما هو موضح فى شكل ٧ – ٣ .




شكل ٧ – ٣

نستمر فى إنتاج زوج من نقاط البحث فى مركز كل منطقة جديدة ومقارنة قيم y المناظرة ، وحذف جزء من منطقة البحث ، حتى تصبح منطقة البحث أقل من 3*sep . وعند حدوث ذلك ، فلن نستطيع أن نميز النقاط الداخلية من حدود منطقة البحث . وعند ذلك ينتهى البحث .

وفى كل مرة تحدث مقارنة بين yr , yl نحذف منطقة البحث التى تحتوى على أصغر قيمة من قيم y . فإذا ماحدث أن كانت قيمتا yr , yl متساويتين ( الشئ الذى يمكن أن يحدث ، إلا أن حدوثه غير عادى ) تتوقف عملية البحث ، حيث يفترض حدوث أكبر قيمة للدالة عند هذه النقطة التى تقع بين yr , yl .

وعند الانتهاء من البحث سواء أكان ذلك لصغر منطقة البحث صغرا كافيا أم لتساوى قيمتى yr , yl ، يمكننا حساب الموقع التقريبى لأكبر قيمة كمايلى :

```
xmax = .5*(xl + xr)
```

والقيمة القصوى المناظرة للدالة يمكن الحصول عليها كمايلى : xmax cos (xmax)

دعنا نعتبر تخطيطا لبرنامج لحالة عامة ، حيث b , a هى كميات مدخلات ، و sep لها قيمة ثابتة هى 0.0001

(١) تحدد قيمة ثابتة للمتغير sep بأنها sep = 0.0001 .

(٢) تقرأ قيم b , a داخل الكمبيوتر .

(٣) يتكرر مايلى حتى تصبح yl مساوية yr ( حيث تقع القيمة القصوى بينهما ) أو تكون أحدث قيمة للفرق بين a , b (b-a) أقل من أو تساوى 3*sep .

أ) تنتج نقطتان داخليتان xr , xl .

ب) تحسب قيمتان yr , yl مناظرتان لهما ، ويتحدد أيهما أكبر .

ج) نقلل منطقة البحث بحذف الجزء الذى لايحتوى على أكبر قيمة من قيم y .

(٤) تقوم ymax , xmax .

(٥) تكتب قيم ymax , xmax ويتوقف العمل .

ويمكن أن تعد الخطوة رقم ٣ على أنها إجراء . ولعمل ذلك ، دعنا نعرف المتغيرات yr , yl , xr , xl , b , a بأنها مؤشرات متغير . ( والقيم التى تمثلها هذه المؤشرات سوف تتغير أثناء الحسابات ويعاد نقلها للأمام وللخلف بين الإجراء والجزء الرئيسى للبرنامج ) .

```
PROCEDURE reduce( VAR a,b,xl,xr,yl,yr : real);
(* Interval reduction routine *)
BEGIN
   xl := a + 0.5*(b-a-sep);
   xr := xl + sep;
   yl := xl*cos(xl);
   yr := xr*cos(xr);
   IF yl > yr THEN b := xr;   (* retain left interval *)
   IF yr > yl THEN a := xl    (* retain right interval *)
END;
```

لاحظ أن sep لم يوضح بأنه مؤشر داخل الإجراء . وسوف نعرف sep كثابت شامل داخل الجزء الرئيسى للبرنامج ، حيث إن قيمته لن تتغير أبدا ، ولاحظ أيضا استخدام الدالة القياسية cos داخل الإجراء ، كما أننا نستخدم هذه الدالة أيضا فى الجزء الرئيسى من البرنامج . وأخيرا فإننا نشير إلى أن تقليل المنطقة الخاصة بالبحث لن يعمل بصورة مناسبة إذا ماكانت بدلا yr , yl متساويتين ( هذا ممكن ، إلا أنه بعيد الحدوث ) . ويمكن تصحيح هذه المشكلة ببساطة ، وذلك بوضع = < من إحدى العلاقات < .

ومن السهل الآن كتابة البرنامج الكامل كمايلى :

```
PROGRAM maximum1(input,output);

(* THIS PROGRAM FINDS THE MAXIMUM OF A
   FUNCTION WITHIN A SPECIFIED INTERVAL *)

   CONST sep = 0.0001;
   VAR a,b,xl,xr,xmax,yl,yr,ymax : real;

   PROCEDURE reduce( VAR a,b,xl,xr,yl,yr : real);
   (* Interval reduction routine *)
   BEGIN
      xl := a + 0.5*(b-a-sep);
      xr := xl + sep;
      yl := xl*cos(xl);
      yr := xr*cos(xr);
      IF yl > yr THEN b := xr;   (* retain left interval *)
      IF yr > yl THEN a := xl    (* retain right interval *)
   END;

BEGIN   (* main action block *)
   readln(a,b);
   REPEAT
      reduce(a,b,xl,xr,yl,yr)
   UNTIL (yl=yr) OR ( b-a <= 3*sep );
   xmax := 0.5*(xl+xr);
   ymax := xmax*cos(xmax);
   writeln(' xmax = ',xmax:8:6,'   ymax = ',ymax:8:6)
END.
```

وينتج عن تنفيذ البرنامج بقيم a = 0 , b = 3141593 المخرجات التالية

```
xmax = 0.860586    ymax = 0.561096
```

وعلى هذا ... يكون لدينا موقع القيمة القصوى والقيمة القصوى نفسها داخل المنطقة الأصلية المعطاه .

## ٤ – الدوال : 4. FUNCTIONS

الدالة function هى برنامج ذاتى تشبه الإجراء فى العديد من الأشياء . ( تذكر مناقشتنا الموجهة لهذا الموضوع فى قسم ٢ -١٢) . وعلى عكس الإجراء ، تستخدم الدالة – على أية حال – للعودة بقيمة واحدة بسيطة النوع إلى نقطة الإشارة إلى الدالة . وأكثر من هذا ... فيشار إلى الدالة بتحديد اسمها داخل تعبير ، كما لو كانت متغيرا من النوع البسيط . ويمكن أن يلى أسم الدالة مؤشر فعلى واحد أو أكثر محصورا أو محصورين بين قوسين ، ومنفصلين بواسطة فواصل . وفى معظم الحالات تنقل المؤشرات الفعلية معلومات لتقويم المؤشرات داخل الدالة . ويمكن على ذلك أن تكون ثوابت أو متغيرات أو تعبيرات .

### مـــثال (٧-١٥)

افرض أن factorial هو اسم دالة تحسب مضروب بعض الكميات الصحيحة ( تذكر أن مضروب الكمية الصحيحة n معروف بأنه n! = 1X2X..Xn ) . وعلى هذا ... يمكن تقويم العلاقة

$$f = x!/a!(x - a)!$$

كما يلى :

```
f := factorial(x)/(factorial(a)*factorial(x-a));
```

حيث a , x , f هى متغيرات من النوع الصحيح .

لاحظ التشابه بين استخدام هذه الدالة التى يعرفها المستفيد ، واستخدام الدوال القياسية التى سبق ذكرها فى قسم ٢ -١٢.

وتحتوى الدالة نفسها على عنوان للدالة وقسم للدالة . ويكتب عنوان الدالة على النحو التالى :

```
FUNCTION name : type
```

أو إذا ماستخدمت مؤشرات ، فإنه يأخذ الشكل التالى :

```
FUNCTION name(formal parameters) : type
```

ويحدد آخر عنصر type نوع البيانات التى تنتج من الدالة ، وتعود إلى نقطة الإشارة إلى الدالة .

وبصفة عامة ... فإن المؤشرات الرسمية تكون مؤشرات قيمة ، بدلا من كونها مؤشرات متغير . وهذا يسمح بالمؤشرات الفعلية المناظرة بأن تكون ثوابت أو متغيرات أو تعبيرات .( تذكر أن هذه المؤشرات تنتج قيم مدخلات للدالة فقط . أما القيمة الفردية لمخرجات الدالة ، فسوف يمثلها اسم الدالة وليس المؤشر ) .

ومجموعة الدالة تشبه مجموعة الاجراء ، وتحتوى على نفس قواعد المدى مثل الإجراء . وداخل مجموعة الدالة يجب أن تحدد قيمة من نوع مناسب ( كما هو محدد فى عنوان الدالة ) للمعرف الذى يمثل اسم الدالة . وهذه هى القيمة التى تعود بها الدالة إلى نقطة الإشارة إليها . ويمكن تحديد قيم لاسم الدالة عند نقطتين أو أكثر داخل مجموعة الدالة وعندما يتم التحديد فلايمكن تغييره .

## مــثال (٧-١٦)

فيما يلى دالة اسمها factorial تحسب مضروب كمية صحيحة :

```
FUNCTION factorial(n : integer) : integer;
(* Calculate the factorial of n *)
VAR factor,product : integer;
BEGIN
   IF n <= 1 THEN factorial
         ELSE BEGIN
                product := 1;
                FOR factor := 2 TO n DO
                   product := product*factor;
                factorial := product
             END
END;
```

أول سطر يحتوى على الكلمة الأساسية FUNCTION هو عنوان الدالة . لاحظ أن العنوان يحتوى على توضيح لمؤشر قيمة . لاحظ أيضا آخر عنصر فى السطر integer ، والذى يحدد أن الدالة سوف تعود بكمية صحيحة النوع .

تقبل هذه الدالة قيمة من قيم n ، ثم تحسب قيمة مضروب n باستخدام متغيرين صحيحين محليين هما product , factor ، وتحدد النتيجة النهائية للمعرف factorial وهو اسم الدالة نفسها .

لاحظ أن هناك تحديدين مختلفين لـ factorial ، إلا أن أحدهما فقط هو الذى يستخدم عند تنفيذ الدالة . ويعتمد الاختيار على القيمة التى تحدد لـ n ، وبمجرد تحديد قيمة factorial ، لا تتغير هذه القيمة داخل الدالة .

ولتوضيح هذه النقطة الأخيرة اعتبر الشكل المختلف التالى للدالة السابقة .

```
FUNCTION factorial(n : integer) : integer;
(* Calculate the factorial of n *)
VAR factor : integer;
BEGIN
   factorial := 1;
   IF n > 1 THEN FOR factor := 2 TO n DO
                    factorial := factorial*factor
END;
```

على السطح تبدو هذه الصيغة أكثر وضوحا عن الصيغة الأصلية ، حيث إنها أبسط . وهذه الصيغة ليست صحيحة على أية حال ، أن قيمة factorial تتغير بعد التحديد الأولى عندما تكون n أكبر من 1 .

وتوضح الدوال بنفس الطريقة التى توضح بها الإجراءات داخل الجزء المنادى ، أو فى أى جزء خارجى محيط بالجزء المنادى . وليس هناك ترتيب خاص بالنسبة للدوال والإجراءات . ( تذكر أنه يجب أن تكون توضيحات الدوال والإجراءات هى آخر توضيحات داخل المجموعة ، وتأتى بعد أى توضيحات لعناوين أو ثوابت أو أنواع أو متغيرات ) .

## مـثال (٧-١٧)

*حساب المضروب* . فيما يلى برنامج بسكال كاملاً ، يحدد مضروب كمية مدخلات معطاه .

```
PROGRAM factorials2(input,output);

(* THIS PROGRAM USES A FUNCTION TO CALCULATE
   THE FACTORIAL OF A GIVEN INTEGER QUANTITY *)

VAR x : integer;

    FUNCTION factorial(n : integer) : integer;
    (* calculate the factorial of n *)
    VAR factor,product : integer;
    BEGIN
       IF n <= 1 THEN factorial := 1
                 ELSE BEGIN
                         product := 1;
                         FOR factor := 2 TO n DO
                            product := product*factor
                         factorial := product
                      END
    END;

BEGIN   (* main action block *)
   write(' Enter a positive integer: ');
   readln(x);
   writeln;
   writeln(' x= ',x,'   x! = ',factorial(x))
END.
```

لاحظ أنه يتم الاتصال بالدالة factorial مرة واحدة فقط فى عبارة writeln داخل الجزء الرئيـسى من البرنامج .

عند تنفيذ هذا البرنامج يجب أخذ الاحتياط للقيمة المحددة لـ x ، حيث إنه من الممكن حدوث سريان زائد إذا ماحددت قيمة كبيرة جدا لـ x . ( تذكر أن المضروب يتزايد بسرعة جدا فى قيمته ) . وهذا صحيح بصفة خاصة بالنسبة لمعظم أجهزة الكمبيوتر الصغيرة ، حيث تكون أكبر كمية صحيحة مسموح بها صغيرة نسبيا .

يمكن للدوال أن تتداخل واحدة داخل الأخرى بنفس طريقة تداخل الإجراءات . وأكثر من هذا ... يمكن أن يحدث تبادل فى تداخل الدوال والإجراءات . وعلى هذا ... يمكن توضيح الدالة داخل إجراء يكون قد سبق توضيحه داخل دالة أخرى ، وهكذا .

## مـثال (٧-١٨)

*محاكاة مباراة الفرصة* Simulation of a Game of Chance . فيما يلى برنامجا ممتعا لمشكلة تستخدم إجراء مع دوال متداخلة .

اللعبة هى لعبة نرد ، يتم فيها قذف زوج من النرد مرة واحدة أو أكثر ، حتى يحدث مكسب أو خسارة للشخص الذى يلقى النرد . ويمكن استخدام الكمبيوتر لأداء المباراة عن طريق تعويض إنتاج الأعداد العشوائية للقذف الفعلى للنرد .

وهناك طريقتان للكسب فى هذه اللعبة . يمكنك أن تلقى زوج النرد مرة واحدة ؛ وتحصل على 7 أو 11 ، أو يمكنك الحصول على 4 أو 5 أو 6 أو 8 أو 9 أو 10 فى أول مرة لإلقاء النرد ، وبعد ذلك فإنك تكرر الحصول على نفس الرقم فى إلقاء النرد مرة تالية قبل أن تحصل على الرقم 7 . وهناك طريقتان أيضا للخسارة . يمكنك أن تلقى النرد مرة وتحصل على 2 أو 3 أو 12 ، أو يمكنك أن تحصل على 4 أو 5 أو 6 أو 8 أو 9 أو 10 فى أول مرة لإلقاء النرد ، ثم تحصل بعد ذلك على 7 فى المحاولة التالية قبل أن تكرر الإلقاء فى محاولة للحصول على الرقم الأصلى .

دعنا نستخدم الكمبيوتر فى أداء هذه المباراة بطريقة تقليدية ، بحيث يمكن محاكاة الشخص الذى يلقى النرد فى كل مرة تضغط على مفتاح return على لوحة مفاتيح الكمبيوتر . عند ذلك تظهر رسالة على الشاشة تحدد ناتج كل عملية لإلقاء النرد . وفى نهاية كل مباراة سوف يسألك الكمبيوتر ما إذا كنت تريد أن تستمر فى اللعب أم لا .

يتطلب البرنامج منتج أعداد عشوائية لإنتاج أعداد عشوائية موزعة توزيعا منتظما تقع بين الصفر و 1 . ( ونعنى بالتوزيع المنتظم uniformly distributed أن أى عدد يقع بين 0 , 1 يمكن أن يظهر بنفس الفرصة التى يظهر بها أى عدد آخر يقع فى نفس المنطقة ) . وبعض صيغ البسكال يوجد بها منتج أعداد عشوائية كدالة قياسية أو كإجراء قياسى ، إلا أن هذا لايتواجد مع كل الصيغ . وعلى هذا ... فسوف نقدم منتج أعداد عشوائية ، مستخدمين أسلوباً معروفاً جيدا ، وهو طريقة power - residue method .

والأعداد التى تنتجها ليست فى واقع الأمر عشوائية ، وذلك نظرا لأن نفس تسلسل الأعداد ينتج دائما لنفس القيمة الأبتدائية ( أى لنفس البذرة seed ) . وعلى أية حال ... فسوف يظهر تسلسل الأعداد على أنه عشوائيا ، وسوف يكون له العديد من خواص الأعداد العشوائية الفعلية .

ولن نناقش أساس منتج الأعداد العشوائية ، حيث إن ذلك لايقع فى إطار محتويات هذا الكتاب ، إلا أن جزء البرنامج الفعلى بسيط . فنحن نقدم ببساطة منتج الأعداد العشوائية فى صيغة دالة تسمى rand . وتكتب هذه الدالة خصيصا للكمبيوتر الذى يستخدم كميات صحيحة تكتب فى 16 بت . ( ومعظم أجهزة الميكروكمبيوتر تقع داخل هذه الفئة ) .

```
FUNCTION rand : real;
(* 16-bit random number generator *)
VAR y,z : integer;
BEGIN
   z := 259*x;
   IF z >= 0 THEN y := z
           ELSE BEGIN y := z + 32767; y := y + 1 END;
   rand := 0.3051757E-4*y;
   x := y
END;
```

وحتى يمكن استخدام هذه الدالة ، يجب أن تحدد قيمة صحيحة لمؤشر المتغير x فى كل مرة تستدعى فيها الدالة . وفى البداية تكون هذه هى البذرة seed التى يمكن أن تحدد كمؤشر مدخلات أو كثابت . والإشارات التالية للدالة rand سوف تستخدم القيمة السابقة لـ y كقيمة جديدة لـ x ( لاحظ أن هذه السمة موجودة فى نهاية الدالة ) .

دعنا نعرف دالة أخرى تسمى throw لمحاكاة إلقاء النرد مرة واحدة . تحتوى هذه الدالة على منتج أعداد عشوائية . سوف يحاكى كل نرد على حدة بإنتاج رقم صحيح عشوائى يأخذ القيمة 1 , 2 , 3 , 4 , 5 , 6 بنفس درجة احتمال حدوث كل رقم من هذه الأرقام ، ثم تضاف القيمتان الخاصتان بكل نرد معا لتمثلا إلقاء زوج النرد مرة واحدة ، ويمكن تمثيل هذه الدالة بالشفرة الشبيهة على النحو التالى : (أنظر الصفحة التالية)

```
FUNCTION throw : integer;
(* Throw the dice one time *)
VAR d1,d2 : integer;

(* define the random number generator *)

BEGIN
   d1 := 1 + trunc(6*rand);
   d2 := 1 + trunc(6*rand);
   throw := d1 + d2
END;
```

وتحتاج هذه الدالة إلى بعض التوضيح . اعتبر على سبيل المثال أول عبارة تحديد ( ... =: d1 ) . الدالة rand (x) تعود برقم عشوائى يقع بين الحدين 0 , 1 ( بين 0 و .... 0.999999 بالفعل ) . وعلى هذا ... فإن التعبير 6*rand (x) يعود بقيمة عدد عشوائى يقع بين 0 , 6 ( بين 0 و .... 5.999999 بالفعل ) ، والتعبير trunc (6*rand (x) ) يعود برقم صحيح عشوائى ، يمكن أن تكون قيمته 0 , 1 , 2 , 3 , 4 , 5 بنفس درجة احتمال حدوث أى منها . وبعد ذلك نضيف 1 لنحصل على الرقم العشوائى الذى يمكن أن يكون 1 , 2 , 3 , 4 , 5 , 6 بنفس درجة احتمال حدوث أى منها . وعلى هذا ... فإن هذا التعبير يحاكى إلقاء النرد .

دعنا نعرف الآن إجراء يسمى Play ، يمكنه أن يحاكى مباراة كاملة ( أى إلقاء النرد أى عدد لازم للمكسب أو للخسارة ) . وعلى هذا ... يحتوى ذلك الإجراء على دالة throw . ويجب أن يكون داخل هذا الإجراء كل قواعد اللعبة . ويمكننا أن نكتب هذه العمليات بالشفرة الشبيهة على النحو التالى :

```
PROCEDURE play;
(* Simulate a single game of craps *)
VAR score,tag : integer;

(* define function throw *)

BEGIN

   (* instruct the user to throw the dice *)

   score := throw;
   CASE score OF
      7,11 : BEGIN   (* win on first throw *)

               (* write a message indicating a win on the first throw *)

             END;
      2,3,12 : BEGIN   (* loss on first throw *)

                 (* write a message indicating a loss on the first throw *)

               END;
      4,5,6,8,9,10 : BEGIN   (* multiple throws *)
                        tag := score;
                        REPEAT
```

(تكملة البرنامج فى الصفحة التالية)

```
                    (* instruct the user to throw the dice again *)

                       score := throw;
                    UNTIL (score = tag) OR (score = 7);
                    IF score = tag
                       THEN BEGIN

                            (* write a message indicating a win *)

                            END
                       ELSE BEGIN

                            (* write a message indicating a loss *)

                            END
                 END   (* multiple throws *)
   END   (* case *)
END;
```

أخيرا يستخدم الجزء الرئيسى من البرنامج للتحكم فى تنفيذ المباراة . وسوف يحتوى هذا الجزء على القليل من المدخلات والمخرجات المتداخلة واستدعاء الإجراء play . ( لاحظ على أية حال أن play سوف يعرف داخل هذا الجزء) . وعلى هذا ... يمكننا كتابة الشفرة الشبيهة على النحو التالى :

```
PROGRAM craps(input,output);
(* Interactive simulation of a game of craps *)
CONST seed = 12345;
VAR count,x : integer;
    n : real;
    answer : char;
    flag : boolean;

(* define procedure play *)

BEGIN
   x := seed;
   flag := true;

   (* generate a welcoming message *)

   WHILE flag DO
      BEGIN
         play;
         write(' Do you want to play AGAIN? (Y/N) ');
         readln(answer);
         IF (answer = 'N') OR (answer = 'n') THEN flag := false
      END;

   (* generate a sign-off message *)

END.
```

لاحظ أن البرنامج سوف يتكرر تنفيذه حتى يحدد اللاعب رغبته فى إيقافه . وفيما يلى برنامج بسكال كاملاً لهذه المباراة ، وبه بعض التعليقات الإضافية :

```
PROGRAM craps(input,output);

(* THIS PROGRAM USES SEVERAL FUNCTIONS TO SIMULATE A GAME OF CRAPS.

  IT INCLUDES A RANDOM NUMBER GENERATOR THAT IS SPECIFICALLY DESIGNED
  FOR A COMPUTER THAT UTILIZES 16-BIT (2-BYTE) INTEGER QUANTITIES. *)

CONST seed = 12345;
VAR count,x : integer;
    n : real;
    answer : char;
    flag : boolean;

   PROCEDURE play;
   (* This procedure simulates a single game of craps *)
   VAR score,tag : integer;

        FUNCTION throw : integer;
        (* This function simulates one throw of a pair of dice *)
        VAR d1,d2 : integer;

             FUNCTION rand : real;
             (* Note: This function can only be used with a computer that
                     supports 16-bit (2-byte) integer quantities. *)
             VAR y,z : integer;
             BEGIN   (* generate a random number *)
                z := 259*x;
                IF z >= 0 THEN y := z
                          ELSE BEGIN y := z + 32767; y := y + 1 END;
                rand := 0.3051757E-4*y;
                x := y
             END;   (* rand *)

        BEGIN   (* throw the dice once *)
           d1 := 1 + trunc(6*rand);
           d2 := 1 + trunc(6*rand);
           throw := d1 + d2
        END;   (* throw *)

        BEGIN   (* play one game *)
           writeln;
           writeln(' Throw the dice . . .');
           readln;
           score := throw;
           CASE score OF
              7,11 : BEGIN   (* win on first throw *)
                       write(score:3,' - Congratulations!');
                       writeln('  You WIN on the first throw!');
                       writeln
                    END;
              2,3,12 : BEGIN   (* loss on first throw *)
                         write(score:3,' - Tough luck!');
                         writeln('  You LOSE on the first throw!');
                         writeln
                      END;
```

(تكملة البرنامج فى الصفحة التالية)

```
4,5,6,8,9,10 : BEGIN   (* multiple throws *)
                  tag := score;
                  REPEAT
                     write(score:3);
                     writeln(' - Throw the dice again . . .');
                     readln;
                     score := throw;
                  UNTIL (score = tag) OR (score = 7);
                  IF score = tag
                     THEN BEGIN
                            write(score:3);
                            write(' - You WIN by matching');
                            writeln(' your first score');
                            writeln
                          END
                     ELSE BEGIN
                            write(score:3);
                            write(' - You LOSE by failing to');
                            writeln(' match your first score');
                            writeln
                          END
                END   (* multiple throws *)
      END   (* case *)
   END;   (* play *)

BEGIN   (* executive routine *)
   x := seed;
   flag := true;
   page;
   writeln(' Welcome to the Game of CRAPS!');
   writeln;
   write(' Press the CARRIAGE RETURN to begin playing');
   readln;
   WHILE flag DO
      BEGIN
         play;
         write(' Do you want to play AGAIN? (Y/N) ');
         readln(answer);
         IF (answer = 'N') OR (answer = 'n') THEN flag := false
      END;
   writeln;
   writeln(' Bye, come back again!')
END.
```

يجب أن يذكر أن منتج الأعداد العشوائية ( الدالة rand ) مصمم لإنتاج سريان زائد لأرقام . وقد تكون هذه مشكلة لبعض المترجمات التى يمكن أن تنتج رسائل أخطاء ، وتفصل البرنامج دون إتمام تنفيذه إذا ماحدث مثل هذا السريان الزائد . وعادة مايمكن التغلب على هذه المشكلة بإضافة أمر خاص ، يجعل المترجم يتأكد من ضغط هذا السريان الزائد . فاذا مانفذ هذا البرنامج – على سبيل المثال – على كمبيوتر شخصى من طراز IBM باستخدام مترجم بسكال IBM قياسى ( المعد من قبل Microsoft ) فيجب إضافة إحدى التعليمات ، وهى MATHCK $ كتعليق . وعلى هذا ... تظهر rand كمايلى :

(أنظر الصفحة التالية)

```
FUNCTION rand : real;
(*$MATHCK- *)
(* Note: This function can only be used with a computer that
      supports 16-bit (2-byte) integer quantities. *)
VAR y,z : integer;
BEGIN   (* generate a random number *)
   z := 259*x;
   IF z >= 0 THEN y := z
         ELSE BEGIN y := z + 32767; y := y + 1 END;
   rand := 0.3051757E-4*y;
   x := y
END;
```

وتشغيل هذا البرنامج فى وسط متداخل مثل الكمبيوتر الشخصى يعطى تعليمات كثيرة . وفيما يلى مجموعة مخرجات تقليدية من هذا البرنامج . وقد وضع خط تحت استجابة المستفيد .

```
Welcome to the Game of CRAPS!

Press the CARRIAGE RETURN to begin playing (press carriage return)

Throw the dice . . .

 7 - Congratulations!  You WIN on the first throw!

Do you want to play AGAIN? (Y/N) y

Throw the dice . . .

 5 - Throw the dice again . . .

 6 - Throw the dice again . . .

12 - Throw the dice again . . .

 7 - You LOSE by failing to match your first score

Do you want to play AGAIN? (Y/N) y

Throw the dice . . .

 9 - Throw the dice again . . .

 4 - Throw the dice again . . .

 9 - You WIN by matching your first score

Do you want to play AGAIN? (Y/N) y

Throw the dice . . .

 2 - Tough luck!  You LOSE on the first throw!

Do you want to play AGAIN? (Y/N) n

Bye, come back again!
```

وأخيرا يجب أن يكون القارئ حذرا من ناحية تغيير قيمة المتغير الشامل أو مؤشر المتغير داخل الدالة ، حيث إن هذا يمكن أن يؤدى إلى نتائج غير متوقعة وغير مطلوبة خارج الدالة . وأحيانا يشار إلى الأنشطة من هذا النوع بأنها تأثيرات جانبية Side effects للدوال . ( لاحظ أن مثل هذه التأثيرات الجانبية غير المطلوبة يمكن أن تنتج عن إجراء أيضا . والتأثيرات الجانبية عادة ماتكون أكثر خطورة مع الدوال – على أية حال – بسبب ميل المبرمجين إلى التفكير فى الدالة بأنها تعود بقيمة واحدة فقط ) .

فمثلا الدالة rand فى مثال ٧ –١٨ تغير من قيمة مؤشر المتغير x فى كل مرة تستدعى فيها الدالة ، وينتج عن ذلك تحديد قيمة مختلفة للمتغير x فى الجزء الرئيسى للبرنامج . وهذا مطلوب طبيعياً لاعادة منتج الأعداد العشوائية . وبصفة عامة ... تجنب هذه العملية كلما كان ذلك ممكنا .

## ٥ – المزيد عن المؤشرات : 5. MORE ABOUT PARAMETERS

فى بعض الأحيان يكون من المرغوب فيه أن يتصل إجراء معين أو دالة معينة بإجراء آخر ، أو بدالة أخرى يكون سبق تعريفه أو تعريفها ( توضيحه أو توضيحها ) خارج مدى الإجراء الذى يجرى الاتصال ، أو الدالة التى تجرى الاتصال . فقد نرغب – على سبيل المثال – فى جعل الإجراء A يستخدم الدالة B ، بالرغم من أن الدالة B معرفة خارج الإجراء A . ويمكن أن يتم ذلك عن طريق عبور الإجراء الخارجى أو الأدلة الخارجية ( أى الدالة B فى حالتنا هذه ) إلى إجراء معين أو دالة معينة ( أى الإجراء A فى حالتنا هذه ) كمؤشر . وعلى هذا ... تقدم لغة البسكال مؤشرات إجراء ومؤشرات دالة ، كما تقدم مؤشرات قيمة ومؤشرات متغير .

ويكتب توضيح المؤشر الرسمى لمؤشر إجراء أو لمؤشر دالة على هيئة عنوان إجراء ، أو عنوان دالة كما هو موضح أدناه .

### مثال (٧–١٩)

افرض أن عملية الإجراء يجب أن تتصل بدالة حقيقية معرفة خارجيا ، وتحتوى على مؤشر من النوع الصحيح . دعنا نشير إلى هذه الدالة بأنها f . يمكن أن يبدو الإجراء ( العملية ) على النحو التالى :

```
PROCEDURE process (FUNCTION f(u : integer) : real; c1,c2 : integer);
VAR c : integer;
    x : real;
BEGIN
   FOR c := c1 TO c2 DO
      BEGIN
         x := f(c);
         writeln(' x=',x)
      END
END;
```

ويمكن أن تحتوى المجموعة الرئيسية للبرنامج على الإشارة التالية لهذا الإجراء

```
process(calc,1,100);
```

حيث الدالة calc تكون معرفة داخل المجموعة الرئيسية . وفيما يلى التكوين الهيكلى لمحتويات المجموعة الرئيسية .

```
PROGRAM main(input,output);
   .
   .
FUNCTION calc(w : integer) : real;
   .
   .
BEGIN   (* function calc *)
   .
   .
   calc := . . .
END;   (* calc *)
PROCEDURE process (FUNCTION f(u : integer) : real; c1,c2 : integer);
VAR c : integer;
    x : real;
BEGIN
   FOR c := c1 TO c2 DO
      BEGIN
         x := f(c);
         writeln(' x=',x)
      END
END;   (* process *)
BEGIN   (* main block *)
   .
   .
   process(calc,1,100);
   .
   .
END.
```

يجب أن تناظر مؤشرات الإجراء الرسمية ، ومؤشرات الدالة الرسمية ، ومؤشرات الإجراء الفعلية ، ومؤشرات الدالة الفعلية مؤشراتها الخاصة . هذا التناظر يجب أن يحتوى على عدد المؤشرات وفئتها ونوعها .

وتكون مؤشرات الإجراء ومؤشرات الدالة مفيدة بصفة خاصة عندما يتصل إجراء محدد ، أو دالة محددة بإجراءات أو دوال مختلفة different ( أى عند استخدام مؤشرات فعلية مختلفة ) عند نقاط استدعاء مختلفة واستخدام هذه الطريقة موضح فى المثالين التاليين .

## مــثال (٧-٢٠)

اعتبر مرة أخرى عملية الإجراء المعرفة فى المثال السابق . يمكن الاتصال بهذا الإجراء عدة مرات فى المجموعة الرئيسية ، وذلك عن طريق مؤشرات دالة مختلفة موجودة فى كل اتصال . وعلى هذا ... قد تشمل المجموعة الرئيسية العبارات التالية :

```
process(calc,1,100);
   .
   .
   .
process(flag,-10,50);
```

حيث الدالتان flag , calc معرفتان داخل المجموعة الرئيسية .

### مـثال (٧-٢١)

فيما يلى مثالاً لدالة تمر إلى دالة أخرى :

```
FUNCTION sum (FUNCTION f(x : real) : real; c1,c2 : integer) : real;
VAR c : integer;
    s : real;
BEGIN
   x := 0;
   FOR c := c1 TO c2 DO x := x + f(0.01*c);
   sum := x
END;
```

فى هذا المثال تحسب الدالة sum مجموع العديد من القيم ، حيث تتحدد كل قيمة عن طريق الاتصال بالدالة f التى تمر إلى sum كمؤشر .

ويمكن أن تحتوى المجموعة الرئيسية على العديد من العبارات ، مثل مايلى :

```
value1 := sum(root(x),a,b);
    .
    .
    .
value2 := sum(cube(x),1,n+1);
```

حيث cube و root هى دوال داخل المجموعة الرئيسية ، n , b , a هى متغيرات صحيحة النوع شاملة . لاحظ أن sum يقبل مؤشرات فعلية مختلفة ( أولا root ثم cube ) فى كل مرة يتم الاتصال به .

ولايسمح بسكال ISO القياسى بمرور الاجراءات القياسية أو الدوال الرئيسية كمؤشرات . وعلى أية حال ... فهناك بعض صيغ اللغة التى لاتضع هذا القيد .

### مـثال (٧-٢٢)

*البحث عن أكبر قيمة* . دعنا نعتبر مشكلة تعظيم التعبير التالى

$$y = x \cos(x)$$

مرة أخرى داخل المنطقة من $x = 0$ وحتى $x = \pi$ كما سبق ذكره فى مثال ٧ -١٤دعنا نستخدم الآن دالة لتقويم هذا التعبير .

وفيما يلى صيغة لبرنامج تمر فيه الدالة max إلى الاجراء reduce كمؤشر .

```
PROGRAM maximum2(input,output);

(* THIS PROGRAM FINDS THE MAXIMUM OF A
   FUNCTION WITHIN A SPECIFIED INTERVAL *)

CONST sep = 0.0001;
VAR a,b,xl,xr,xmax,yl,yr,ymax : real;

    FUNCTION curve(x : real) : real;
    (* Define the function f=x*cos(x) *)
    BEGIN
       curve := x*cos(x)
    END;
```

(تكملة البرنامج فى الصفحة التالية)

```
      PROCEDURE reduce( FUNCTION f(x : real) : real;
                             VAR a,b,xl,xr,yl,yr : real);
      (* Interval reduction routine *)
      BEGIN
         xl := a + 0.5*(b-a-sep);
         xr := xl + sep;
         yl := f(xl);
         yr := f(xr);
         IF yl > yr THEN b := xr;   (* retain left interval *)
         IF yr > yl THEN a := xl    (* retain right interval *)
      END;

   BEGIN   (* main action block *)
      readln(a,b);
      REPEAT
         reduce(curve,a,b,xl,xr,yl,yr);
      UNTIL (yl=yr) OR ( b-a <= 3*sep )
      xmax := 0.5*(xl+xr);
      ymax := curve(xmax);
      writeln(' xmax = ',xmax:8:6,'   ymax = ',ymax:8:6)
   END.
```

وينتج عن تنفيذ هذا البرنامج نفس المخرجات الموضحة فى مثال ٧ -١٤ ويجب أن يكون مفهوما – على أية حال – أنه ليس هناك ميزة خاصة فى مرور الدالة إلى الإجراء فى هذا المثال ، حيث إن الإجراء يستخدم نفس الدالة فى كل مرة يتم الاتصال به . وعلى هذا ... تكون الطريقة الأكثر وضوحا للاتصال بالدالة دون without تركها والذهاب إلى الإجراء كمؤشر . وفيما يلى صيغة لبرنامج يفعل ذلك .

```
   PROGRAM maximum3(input,output);

   (* THIS PROGRAM FINDS THE MAXIMUM OF A
      FUNCTION WITHIN A SPECIFIED INTERVAL *)

   CONST sep = 0.0001;
   VAR a,b,xl,xr,xmax,yl,yr,ymax : real;

      FUNCTION curve(x : real) : real;
      (* Define the function f=x*cos(x) *)
      BEGIN
         curve := x*cos(x)
      END;

      PROCEDURE reduce( VAR a,b,xl,xr,yl,yr : real);
      (* Interval reduction routine *)
      BEGIN
         xl := a + 0.5*(b-a-sep);
         xr := xl + sep;
         yl := curve(xl);
         yr := curve(xr);
         IF yl > yr THEN b := xr;   (* retain left interval *)
         IF yr > yl THEN a := xl    (* retain right interval *)
      END;

   BEGIN   (* main action block *)
      readln(a,b);
      REPEAT
```

(تكملة البرنامج فى الصفحة التالية)

```
      reduce(a,b,xl,xr,yl,yr)
   UNTIL (yl=yr) OR ( b-a <= 3*sep );
   xmax := 0.5*(xl+xr);
   ymax := curve(xmax);
   writeln(' xmax = ',xmax:8:6,'   ymax = ',ymax:8:6)
END.
```

وينتج عن تنفيذ هذا البرنامج النتائج الصحيحة المحسوبة المسماه :

```
xmax = 0.860586    ymax = 0.561096
```

لقد رأينا الآن ثلاث طرق مختلفة لحل هذه المشكلة . والسؤال أى هذه الطرق أفضل هو سؤال تعتمد اجابته على المبرمج ذلك بالرغم من أن آخر صيغة تبدو مناسبة لهذه المشكلة الخاصة .

## ٦ - الإعادة الذاتية : 6. RECURSION

إحدى السمات الجيدة للبسكال هى امكانية استدعاء الإجراء نفسه ، وإمكانية استدعاء الدالة نفسها . وهذا مايعرف بالإعادة الذاتية . واستخدام الإعادة الذاتية مريح بصفة خاصة بالنسبة للمشاكل التى يمكن تعريفها باصطلاحات إعادة طبيعية ( بالرغم من أن مثل هذه المشاكل يمكن أن تبرمج دون استخدام إعادة ذاتية أيضا ) .

عند كتابة إجراء إعادة ذاتية أو دالة إعادة ذاتية ، فمن الضرورى أن يحتوى هذا الإجراء ( أو هذه الدالة ) على شرط للإنهاء . وهذا يمنع الإعادة الذاتية من الاستمرار بصورة دائمة . وطالما أن شرط الإنهاء لم يتحقق ، فيقوم الإجراء ( أو الدالة ) باستدعاء نفسه فى المكان المناسب ، كما لو كان يستدعى أى إجراء آخر ، أو أى دالة أخرى .

### مثال (٧-٢٣)

*حساب المضروبات* . فى مثال ٧ -١٧رأينا برنامجا لحساب مضروب كمية مدخلات معطاه باستخدام دالة غير معادة ذاتيا فى أداء الحسابات الفعلية . دعنا نرى الآن كيف يمكن أداء هذه الحسابات باستخدام الإعادة الذاتية .

لاحظ أولا أنه يمكن تعريف مضروب كمية صحيحة موجبة n عن طريق الإعادة الذاتية بأنها

$$n! = n \times (n-1)!$$

حيث $1! = 1$

وهذه المعادلات تقدم الأساس لدالة الإعادة الذاتية التالية :

```
FUNCTION factorial (n : integer) : integer;
(* Calculate the factorial of n *)
BEGIN
   IF n <= 1 THEN factorial := 1
            ELSE factorial := n*factorial(n-1)
END;
```

لاحظ أن هذه الدالة أبسط كثيرا عن الدالة المقدمة فى مثال ٧ -١٧. والتناظر القريب بين هذه الدالة وتعريف المشكلة الطبيعية بالنسبة للإعادة الذاتية يجب أن يكون ظاهرا . لاحظ أن جزء IF - THEN يقدم شرط إنهاء أيضا ، والذى ينشط عندما يصبح المؤشر a أقل من أو يساوى 1 ( n لايمكن أن تصبح أقل من 1 إلا إذا كانت قيمتها الأصلية أقل من 1 ) .

وفيما يلى برنامج بسكال كاملاً . والمجموعة الرئيسية تشبه جدا نظيرتها الموجودة فى مثال ٧ -١٧.

```
PROGRAM factorials3(input,output);

(* THIS PROGRAM USES A RECURSIVE FUNCTION TO CALCULATE
          THE FACTORIAL OF A GIVEN INTEGER QUANTITY *)

VAR x : integer;

    FUNCTION factorial (n : integer) : integer;
    (* Calculate the factorial of n *)
    BEGIN
       IF n <= 1 THEN factorial := 1
                 ELSE factorial := n*factorial(n-1)
    END;

BEGIN   (* main action block *)
   write(' Enter a positive integer: ');
   readln(x);
   writeln;
   writeln(' x= ',x,'   x! = ',factorial(x))
END.
```

وعلى هذا ... لحساب مضروب كمية فردية n ، يتم تكرار الاتصال بالدالة factorial مرة فى المجموعة الرئيسية ، وعدد (n-1) من المرات داخل نفسها .

وفى بعض التطبيقات يسمح استخدام الإعادة الذاتية بتخزين عناصر بيانات متعددة داخل الكمبيوتر ، دون الحاجة إلى هيكل بيانات خاص . وهذا موضح فى المثال التالى .

## مـــثال (٧-٢٤)

*الطباعة الخلف* . افرض أننا نريد كتابة برنامجًا يقرأ سطرا من أحد النصوص ، طوله غير محدد ، حتى يحدث نهاية للسطر ، وتعود العربة لبداية السطر الجديد . بعد ذلك يكتب السطر مطبوعا بترتيب معكوس ( أى طباعة خلفية ) .

الطريقة التقليدية لهذه المشكلة هى عن طريق تخزين الرموز فى قائمة كما قرأت ، ثم تطبع بترتيب عكسى ، وذلك بالاستمرار فى القائمة من الخلف ، وذلك بعد الانتهاء من إدخال كل الرموز . ولأجل عمل ذلك ... يجب أن ننتج هيكل بيانات ( مثل المنظومة array ) لتخزين الرموز فى الصورة التى أدخلت بها . ( سوف نرى كيف يمكن عمل ذلك فى الفصل التاسع من الكتاب ) .

والطريقة الأخرى لحل هذه المشكلة هى استخدام الإعادة الذاتية ، كما هو موضح أدناه :

```
PROGRAM backwards(input,output);

(* THIS PROGRAM READS A LINE OF TEXT AND
   THEN WRITES IT OUT IN REVERSE ORDER. *)

    PROCEDURE flipit;
    (* Read single characters recursively,
                    then write them out. *)
    VAR c : char;

    BEGIN
       read(c);
       IF NOT eoln THEN flipit;
       write(c)
    END;

BEGIN   (* main action block *)
   writeln(' Enter a line of text, then press RETURN');
   writeln;
   flipit
END.
```

والإجراء flipit هو مفتاح البرنامج . فيستدعى هذا الإجراء نفسه باستخدام الإعادة الذاتية ليقرأ فى الكمبيوتر رموزا متتالية حتى ينتهى السطر ( حتى يتحقق eoln ) . لاحظ أن الاستدعاء للإعادة الذاتية يسبق عبارة write . وعلى هذا تقرأ كل all الرموز داخل الكمبيوتر قبل طباعة أى منها خارجه . وبعد قراءة كل الرموز داخل الكمبيوتر . يكتب أحدث رمز تم دخوله خارج الكمبيوتر ( وذلك لأن هذا الرمز يناظر استدعاء للإجراء ) ، يليه الرمز السابق له مباشـــرة وهكذا .

فإذا كان البرنامج ينفذ – على سبيل المثال – بسطر المدخلات التالى :

```
NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR COUNTRY
```

فإن المخرجات المناظرة تصبح كمايلى :

```
YRTNUOC RIEHT FO DIA EHT OT EMOC OT NEM DOOG LLA ROF EMIT EHT SI WON
```

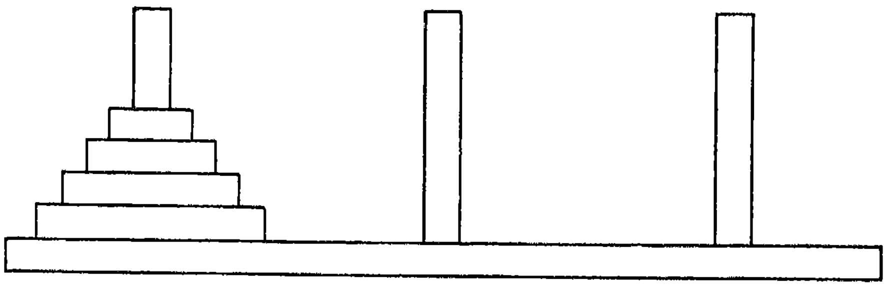
إذا مااستدعى إجراء ( أو دالة ) يحتوى على متغيرات محلية باستخدام الإعادة الذاتية ، فسوف تنتج فئة مختلفة different من المتغيرات المحلية أثناء كل مرة يحدث فيها استدعاء . وتكون أسماء المتغيرات المحلية بالطبع لها نفس الأسماء ( كما سبق توضيحها داخل الإجراء أو الدالة ) . وعلى أية حال ... فإن فئة مختلفة من القيم سوف تصاحب أسماء المتغيرات المحلية هذه فى كل مرة ينشط فيها الإجراء أو الدالة . وتصبح هذه القيم متاحة مع عدم اللف " unwinds " للإجراء أو الدالة ، كما تمت استدعاءات الإعادة الذاتية السابقة .

## مـثال (٧–٢٥)

اعتبر إجراء الإعادة الذاتية المسمى flipit الموضح فى مثال ٧ –٢٤. يحتوى هذا الإجراء على متغير محلى من النوع الصحيح c . وفى كل مرة يستدعى الإجراء ، تتم قراءة رمز مختلف فى الكمبيوتر ويحدد للمتغير c . وعلـــى هـــذا ... يصاحب c العديد من الرموز المختلفة ، وتخزن على التوالى داخل ذاكرة الكمبيوتر . هذه الرموز تكون مســتقلة عن بعضها . وتكتب خارج الكمبيوتر فى الترتيب المناسب عند إتمام استدعاءات الإعادة الذاتية ( والتى تعنى أن أحدث قيمة للمتغير c تم إدخالها وتخزينها وكتابتها خارج الكمبيوتر ) .

## مـثال (٧-٢٦)

*برج هانوى* . فيمايلى مثالاً يستخدم بكثرة لتوضيح كيفية استغلال الإعادة الذاتية فى تبسيط برمجة مشكلة تبدو معقدة . وتهتم المشكلة بلعبة أطفال شائعة الاستخدام ، تحتوى على ثلاثة أعمدة ، وعدد من الأقراص مختلفة الأحجام . وكل قرص به ثقب فى منتصفه لتركب حول الأعمدة . وفى البداية تركب الأقراص على العمود الموجود فى أقصى اليسار ، مرتبة طبقا لحجمها ، على أن تكون الكبرى فى قاعدة العمود ، فالأقل فالأقل ، والصغرى فى نهايته ( انظر شكل ٧-٤ ) .




شكل ٧ - ٤

وهدف المباراة هو نقل الأقراص من العمود الموجود على أقصى اليسار إلى العمود الموجود على أقصى اليمين ، دون رفع قرص اكبر فوق قرص أصغر . ويتم تحريك قرص واحد فقط فى نفس الوقت . ويجب أن يوضع كل قرص فى أحد الأعمدة دائما .

والاستراتيجية العامة هى اعتبار أحد الأعمدة أنه الأصل ، وعمود آخر أنه الغاية . ويستخدم العمود الثالث فى التخزين المرحلى ليسمح بحركة الأقراص . وعلى هذا ... إذا وضع n قرص فى العمود الموجود على أقصى اليسار فى البداية ، فيمكن تمثيل مشكلة نقل هذه الأقراص التى عددها n إلى العمود الأيمن على النحو التالى :

(١) نقل أعلى ( n-1 ) قرص من العمود الأيسر إلى العمود المتوسط باستخدام العمود الأيمن كمخزن مرحلى .

(٢) نقل القرص المتبقى إلى العمود الأيمن .

(٣) نقل ( n-1 ) قرص الموجودة فى العمود المتوسط إلى العمود الأيمن باستخدام العمود الأيسر كمخزن مرحلى .

وعلى هذا ... يمكن تمثيل المشكلة فى صورة الإعادة الذاتية المطبقة مع أى قيمة لـ n أكبر من 0. ( عندما تكون n=1 ، فإننا ننقل قرصاً ببساطة من العمود الأيسر إلى العمود الأيمن ) .

ولكى نكتب برنامجا لهذه المباراة ، فإننا نضع أرقاما أولا للأعمدة ، بحيث إن left = 1 و center = 2 و right = 3 ، ثم نعد بعد ذلك إجراء إعادة ذاتية يسمى transfer ، والذى ينقل n قرصا من عمود إلى عمود آخر . ونشير إلى الأعمدة بمتغيرات صحيحة النوع اسمها origin , destination , other . وعلى هذا... فإذا ماحددنا القيمة 1 للمتغير origin والقيمة 3 للمتغير destination والقيمة 2 للمتغير other ، فإننا نكون فى واقع الأمر محددين حركة n قـرص مـن العـمود 1 ( العمود الموجود على أقصى اليسار ) إلى العمود 3 ( العمود الموجود على أقصى اليمين ) باستخدام العمود 2 ( العمود الموجود فى المنتصف ) فى التخزين المرحلى . وبهذه الملاحظات يكون للإجراء الهيكل التالى :

```
PROCEDURE transfer(n,origin,destination,other : integer);

IF n>0 THEN

(* transfer n-1 disks from the origin to the intermediate pole *)

(* transfer the remaining disk from the origin to the destination *)

(* transfer n-1 disks from the intermediate pole to the destination *)

END;
```

يمكن حدوث نقل (n-1) قرص عن طريق استدعاء الإعادة الذاتية لـ transfer . وعلى هذا ... يمكننا كتابة .

```
transfer(n-1,origin,other,destination)
```

لأول نقل وكتابة

```
transfer(n-1,other,destination,origin)
```

للنقل الثانى . ( لاحظ ترتيب المؤشرات فى كل استدعاء ) . ويتطلب نقل قرص واحد من الأصل إلى الغاية كتابة قيم destination , origin . ويمكن أن يحدث ذلك باستدعاء إجراء آخر اسمه diskmove . وهو معرف داخل transfer ويمكن كتابة إجراء النقل كاملا كمايلى :

```
PROCEDURE transfer(n,origin,destination,other : integer);

   PROCEDURE diskmove(origin,destination : integer);
   BEGIN
      writeln('Move ',origin:1,' to ',destination:1)
   END;

BEGIN
   IF n>0 THEN BEGIN
                 transfer(n-1,origin,other,destination);
                 diskmove(origin,destination);
                 transfer(n-1,other,destination,origin)
              END
END;
```

وأصبح الأمر الآن سهلا ، وذلك بإضافة المجموعة الرئيسية التى تقرأ أساسا قيمة n ، ثم تبدأ الحسابات باستدعاء transfer . وفى أول مرة لاستدعاء الإجراء تحدد المؤشرات الفعلية بأنها أرقام صحيحة . أى أن

```
transfer(n,1,3,2)
```

وهذا يحدد نقل كل الأقراص التى عددها n من العمود 1 ( عمود الأصل ) إلى العمود 3 ( عمود الغاية ) باستخدام عمود 2 فى التخزين المرحلى .

وفيما يلى البرنامج كاملا :

```
PROGRAM towersofhanoi(input,output);

(* THIS PROGRAM SOLVES A WELL-KNOWN GAME
       USING RECURSIVE PROCEDURE CALLS *)

VAR n : integer;

   PROCEDURE transfer(n,origin,destination,other : integer);
   (* Transfer n disks from the origin to the destination *)

      PROCEDURE diskmove(origin,destination : integer);
      (* Move a single disk from the origin to the destination *)
      BEGIN
         writeln('Move ',origin:1,' to ',destination:1)
      END;   (* diskmove *)

   BEGIN   (* transfer *)
      IF n>0 THEN BEGIN
                     transfer(n-1,origin,other,destination);
                     diskmove(origin,destination);
                     transfer(n-1,other,destination,origin)
                  END
   END;   (* transfer *)

BEGIN   (* main action block *)
   write(' Enter the number of disks -> ');
   readln(n);
   writeln;
   transfer(n,1,3,2)
END.
```

ويجب أن يكون مفهوما أن مجموعات القيم المختلفة different تعرف لمؤشرات القيمة , other , destination origin , n فى كل مرة تستدعى فيها transfer . هذه المجموعات من القيم تخزن منفصلة عن بعضها داخل ذاكرة الكمبيوتر . وهذه الإمكانية فى تخزين هذه المجموعات المنفصلة من القيم ، ثم استعادتها فى الوقت المناسب ، هى التى تسمح بعمل إعادة ذاتية .

وعند تنفيذ البرنامج لقيمة n = 3 نحصل على المخرجات التالية . ( لاحظ أن استجابة المستفيد موضوع تحتها خط ) .

```
Enter the number of disks -> 3

Move 1 to 3
Move 1 to 2
Move 3 to 2
Move 1 to 3
Move 2 to 1
Move 2 to 3
Move 1 to 3
```

ويجب أن يفكر القارئ في هذا التسلسل للنقل للتحقق من صحته .

وعلى القارئ دراسة هذا المثال بعناية ، فالمنطق ليس سهلا ، بالرغم من بساطة البرنامج .

فى كل أمثلة الإعادة الذاتية التى رأيناها حتى الآن يقوم إجراء واحد ( أو دالة واحدة ) باستدعاء نفسه . وهناك صيغة أخرى للإعادة الذاتية تسمى إعادة ذاتية مشتركة ، والتى يستدعى إجراء ( أو دالة ) إجراء آخر ، وبعد ذلك يستدعى الإجراء الثانى ( أو الدالة ) الإجراء الأول . تذكر على أية حال أنه لايمكن الاتصال بإجراء ( أو بدالة ) قبل توضيحه . وعلى هذا ... فهناك مشكلة فى الإعادة الذاتية المشتركة عندما يستدعى الإجراء ( الدالة ) الأول الإجراء الثانى قبل أن يكون معرفا .

ويحل البسكال هذه المشكلة بالسماح بتوضيح صورى dummy declaration للإجراء الثانى ( أو الدالة الثانية ) ليسبق الإجراء الأول ( أو الدالة الأولى ) . يحتوى التوضيح الصورى على إجراء ( أو دالة ) بداية يتبعها الكلمة الأساسية forward . أما التوضيح الحقيقى للإجراء ( أو الدالة ) ، فيكتب بعد ذلك فى المكان المناسب بعد توضيح أول إجراء ( أو دالة ) . ويكتب بالطريقة المعتادة ، فيما عدا ان قائمة المؤشر الرسمى تحذف من العنوان . ( وفى حالة الدالة يحذف نوع النتيجة أيضا ) .

### مـثال (٧-٢٧)

فيما يلى تخطيطًا هيكليًا لدالة تستدعى إجراء ، ثم بعد ذلك تستدعى الدالة باستخدام الإعادة الذاتية .

```
FUNCTION second(a,b : real) : real; forward;

PROCEDURE first(x,y : real);
VAR w : real;
BEGIN
   .
   .
   w := second(x,y);
   .
   .
END;

FUNCTION second;
VAR u,v : real;
BEGIN
   .
   .
   first(u,v);
   .
   .
   second := . . .
END;
```

لاحظ أن التوضيح الصورى للمتغير second ضرورى ، بحيث إن first يمكنه الاتصال بـ second قبل أن يتم توضيح second . وبعد ذلك فإن التوضيح الحقيقى لـ second يلى توضيح first . لاحظ – على أية حال – أن عنوان الدالة مختصر الآن .

واخيرا يجب أن يكون مفهوما ان استخدام الإعادة الذاتية ليس ضروريا أن يكون أفضل طريقة لحل المشكلة ، بالرغم من أن تعريف المشكلة قد يكون له طبيعة إعادة ذاتية . فالحل دون استخدام الإعادة الذاتية قد يكون أكثر كفاءة بالنسبة لاستغلال ذاكرة الكمبيوتر ولسرعة التنفيذ . وعلى هذا ... فإن تكرار استخدام الإعادة الذاتية يشتمل على المواءمة بين البساطة وسوء الأداء . ويجب الحكم على ذلك على كل مشكلة طبقا لمزاياها الخاصة لها .

## أسئلة للمراجعة : Review Questions

(١) ماهى مميزات تجزئة البرنامج ؟ ماهى أنواع أجزاء البرنامج المتاحة فى البسكال ؟

(٢) كيف يمكن الإشارة إلى إجراء ؟ وماهى الكلمات الأخرى المستخدمة فى الإشارة إلى دليل إجراء ؟

(٣) صف هيكل قواعد إعداد الإجراء . كيف يكتب عنوان الإجراء ؟

(٤) فى أى مكان من برنامج البسكال يجب أن يظهر توضيح الإجراء ؟

(٥) ماهو الفرق بين المعرفات الشاملة والمعرفات المحلية ؟ وأيهما يفضل استخدامه ، ولماذا ؟

(٦) ماذا يعنى مدى المعرف ؟

(٧) ماذا تعنى الإجراءات المتداخلة ؟ ماهى القيود التى تتطلب اهتماما خاصا عند محاولة الاتصال بإجراءات متداخلة ؟

(٨) هل مفهوم المدى يطبق على المعرفات التى تمثل ثوابت ومتغيرات فقط ؟ وضح ذلك .

(٩) كيف تؤثر قواعد المدى على نقل التحكم الذى يصاحب استخدام عبارة GOTO ؟

(١٠) ماهى المؤشرات ؟ وكيف تستخدم ؟

(١١) ماهو الفرق بين المؤشرات الفعلية والمؤشرات الرسمية ؟

(١٢) لخص القواعد التى يجب اتباعها عند التعويض بمؤشرات فعلية ، بدلا من مؤشرات رسمية .

(١٣) ماهى الأربع فئات الخاصة بالمؤشرات الرسمية ؟ وكيف تختلف عن بعضها ؟

(١٤) ماهى مؤشرات القيمة ؟ كيف تكتب مؤشرات القيمة الرسمية ؟ وفى أى نوع من أنواع التطبيقات تستخدم مؤشرات القيمة بصفة عامة ؟

(١٥) ماهى مؤشرات المتغير ؟ كيف تكتب مؤشرات المتغير الرسمية ؟ وكيف تختلف مؤشرات المتغير عن مؤشرات القيمة بالنسبة لـ :

(أ) طريقة كتابتها .

(ب) طريقة استخدامها .

(١٦) ماهى الدالة ؟ كيف تختلف الدالة عن الإجراء ؟

(١٧) كيف يشار إلى الدالة ؟ قارن ذلك بالإشارة إلى الإجراء .

(١٨) صف هيكل قواعد إعداد الدالة . كيف يكتب عنوان الدالة ؟ وكيف يختلف هيكل القواعد هذا عن نظيره للإجراء ؟

(١٩) أين يجب أن يظهر توضيح الدالة فى برنامج البسكال ؟ وهل هناك ترتيب خاص يجب اتباعه بالنسبة لتوضيحات الدوال وتوضيحات الإجراءات ؟

(٢٠) هل يمكن أن تتداخل الدوال بنفس الطريقة مثل الإجراءات ؟ وهل يمكن تبادل تداخل الإجراءات والدوال ؟

(٢١) ماذا تعنى الآثار الجانبية التى تصاحب الدالة ؟ وهل يصاحب الإجراءات آثار جانبية أيضا ؟

(٢٢) متى يكون مرغوبا فى مرور الإجراء أو الدالة كمؤشر ؟

(٢٣) متى يمر الإجراء أو الدالة كمؤشر ، وماذا يجب أن يحفظ بين المؤشرات الرسمية والمؤشرات الفعلية ؟

(٢٤) كيف تكتب مؤشرات الإجراء الرسمية ومؤشرات الدالة الرسمية ؟

(٢٥) هل يمكن أن يمر الإجراء القياسى أو الدالة القياسية كمؤشر ؟

(٢٦) ماذا تعنى الإعادة الذاتية ؟ وهل هى أساسية فى البرمجة ؟

(٢٧) متى يكتب إجراء أو دالة إعادة ذاتية ، وكيف يمكن منع عملية الإعادة الذاتية من الاستمرار اللانهائى ؟

(٢٨) ماهى المميزات التى تقدمها الإعادة الذاتية بالنسبة لتخزين عناصر بيانات متعددة ؟

(٢٩) افرض أن إجراء ( أو دالة ) إعادة ذاتية يحتوى على مجموعة من المتغيرات المحلية . كيف تفسر المتغيرات المحلية كلما تكرر الاتصال بالإجراء ( أو الدالة ) .

(٣٠) ماذا تعنى الإعادة الذاتية المشتركة ؟

(٣١) متى يعرف إجراء ( أو دالة ) إعادة ذاتية مشتركة ، وماهى المشكلة التى يحلها التوضيح الصورى ؟ وكيف تكتب التوضيحات الصورية ؟ وماهى التغييرات التى يجب إجراؤها على التوضيحات الأخرى بسبب التوضيحات الصورية ؟

## مسائل محلولة : Solved Problems

(٣٢) فيما يلى العديد من تخطيطات دلائل إجراءات ودوال . ( افرض أن sample2 , sample1 , sample هى إجراءات ، وأن demo2 , demo1 , demo هى دوال . التوافقية فى النوع مفروضة خلال التخطيطات ) .

(*a*) `sample;`
(*b*) `sample1(a,b,c);`
(*c*) `sample1(a+b,2*x,sqr(c));`
(*d*) `sample2(sample,demo);`
(*e*) `z := demo;`
(*f*) `z := a + 2*demo - 3;`
(*g*) `z := demo1(a,b,c);`
(*h*) `z := y + demo1(a+b,2*x,sqr(c));`
(*i*) `z := demo2(sample,demo);`
(*j*) `IF demo1(a,b,c) < zstar THEN z := demo1(a,b,c);`

(٣٣) فيما يلى العديد من هياكل توضيحات إجراءات ودوال . ( هذه التوضيحات تناظر الإجراءات والدوال المشار إليها فى المشكلة رقم ٣٢ ) .

(*a*)
```
PROCEDURE sample;
VAR . . .;   (* local variables *)
BEGIN
   .
   .
   .
END;
```

(*b*)
```
PROCEDURE sample1(x,y,z : real);
VAR . . .;   (* local variables *)
BEGIN
   .
   .
   .
END;
```

(*c*)
```
PROCEDURE sample1(VAR x,y,z : real);
VAR . . .;   (* local variables *)
BEGIN
   .
   .
   .
END;
```

( لاحظ أن الجزء (b) يستخدم مؤشرات قيمة ، بينما يستخدم الجزء (c) مؤشرات متغير . مؤشرات المتغير ضرورية إذا ماتغيرت المؤشرات داخل الإجراء ، وعادت القيم الجديدة إلى نقاط الاتصال ) .

(*d*)
```
PROCEDURE sample2(PROCEDURE sample; FUNCTION demo : real);
VAR . . .;   (* local variables *)
BEGIN
   .
   .
   .
END;
```

(*e*)
```
PROCEDURE sample2(PROCEDURE sample1(x,y,z : real);
          FUNCTION demo1(VAR a,b,c : real) : real);
VAR . . .;   (* local variables *)
BEGIN
   .
   .
   .
END;
```

(*f*)
```
FUNCTION demo : real;
VAR . . .;   (* local variables *)
BEGIN
   .
   .
   .
   demo := . . .
END;
```

(*g*)
```
FUNCTION demo1(x,y,z : real) : real;
VAR . . .;   (* local variables *)
BEGIN
   .
   .
   demo1 := . . .
END;
```

```
(h) FUNCTION demo2(PROCEDURE sample; FUNCTION demo : real) : real;
    VAR . . .;   (* local variables *)
    BEGIN
       .
       .
       .
       demo2 := . . .
    END;

(i) FUNCTION demo2(PROCEDURE sample1(VAR x,y,z : real);
             FUNCTION demo1(a,b,c : real) : real) : real;
    VAR . . .;   (* local variables *)
    BEGIN
       .
       .
       .
       demo2 := . . .
    END;
```

(٣٤) الهيكل التالى هو تخطيط يبين توضيح إجراء ( أو دالة ) مع الأدلة المناظرة له ( لها ) .

```
(a) PROGRAM sample1(input,output);
    VAR a,b,c : integer;

       PROCEDURE  proc1(a,b,c : integer);
       VAR . . .;   (* local variables *)
       BEGIN
          .
          .
          .
       END;

       FUNCTION funct1(x,y : integer) : integer;
       VAR r,s,t : integer;
       BEGIN
          .
          .
          proc1(r,s,t);
          .
          .
          funct1 := . . .
       END;

    BEGIN   (* main action statements *)
       .
       .
       c := funct1(3,-1);
       .
       .
       proc1(a,b+c,0);
       .
       .
       proc1(a+b,funct1(0,1),c+3)
    END.
```

(b)
```
PROGRAM sample2(input,output);
VAR a,b,c : integer;

   FUNCTION funct2(x,y : integer) : integer; forward;

   PROCEDURE proc2(a,b,c : integer);
   VAR p,q,r : integer;
   BEGIN
      .
      .
      r := funct2(p,q-1);
      .
      .
   END;

   FUNCTION funct2;
   VAR r,s,t : integer;
   BEGIN
      .
      .
      proc2(r,s,t);
      .
      .
      funct2 := . . .
   END;

BEGIN   (* main action statements *)
   .
   .
   c := funct2(3,-1);
   .
   .
   proc2(a,b+c,0);
   .
   .
   proc2(abs(a+b),c,funct2(1,1))
END.
```

لاحظ أن هذه المثال يستخدم سمة التقدم forward .

(c)
```
PROGRAM sample3(input,output);
VAR a,b,c : integer;

   FUNCTION funct3(p,q : integer) : integer;
   VAR r,s,t : integer;

      PROCEDURE proc3(VAR x,y,z : integer);
      VAR . . .;   (* local variables *)
      BEGIN
         .
         .
         .
      END;
```

```
   BEGIN   (* funct3 action statements *)
      .
      .
      proc3(r,s,t);
      .
      .
      funct3 := . . .
   END;

BEGIN   (* main action statements *)
   .
   .
   c := funct3(1,2);
   .
   .
   a := funct3(2,0);
   .
   .
END;
```

Note that proc3 is embedded within funct3.

(*d*)
```
PROCEDURE factorial (VAR f : integer; n : integer);
BEGIN
   IF n <= 1 THEN f := 1
            ELSE BEGIN
                    factorial (f,n-1);
                    f := n*f
                 END
END;
```

يوضح هذا المثال استخدام إعادة لإجراء ( قارن ذلك مع الدالة المبينة فى مثال ٧ -٣٢. لاحظ أن هذا الإجراء يستخدم كل من مؤشر متغير (f) ومؤشر قيمة (a) .

## Supplementary Problems مشاكل متكاملة :

(٣٥) التخطيطات الهيكلية التالية توضح حالات مختلفة تحتوى على استخدام إجراءات ودوال . بعضها مكتوب بصورة غير صحيحة . عرف الأخطاء .

(*a*)
```
PROCEDURE sum(a,b,c : real);
VAR sum : real;
BEGIN
   sum := a + b + c
END;
```

(*b*)
```
FUNCTION sum(a,b,c : real) : real;
BEGIN
   sum := a + b + c
END;
```

(*c*)
```
PROCEDURE sum1(a,b,c : real; VAR sum : real),
BEGIN
   sum := a + b + c
END;
```

(*d*)
```
FUNCTION sum(x : real);
VAR sum : real;
BEGIN
  sum := sum + x
END;
```

(*e*)
```
PROGRAM sample(input,output);
VAR a,b,c : char;

   PROCEDURE proc1(x,y : integer);
   BEGIN
      .
      .
      .
   END;

  BEGIN   (* main action statements *)
    .
    .
    proc1(a,b);
    .
    .
  END.
```

(*f*)
```
PROGRAM sample(input,output);
VAR a,b : integer;
    flag : boolean;

   PROCEDURE proc1(x,y : integer);
   VAR . . .;   (* local variables *)
   BEGIN
     .
     .
     REPEAT
       .
       .
       IF . . . THEN flag := false;
       .
       .
     UNTIL flag = false
   END;

  BEGIN   (* main action statements *)
    .
    .
    flag := true;
    proc1(a,a+2);
    .
    .
    flag := true;
    proc1(abs(b-a),0);
    .
    .
  END.
```

(*g*)
```
PROGRAM sample(input,output);
VAR a,b,t : integer;

   PROCEDURE proc2(VAR x,y : integer);
   VAR . . .;   (* local variables *)
```

```
     BEGIN
        .
        .
        IF t > 0 THEN y := . . .
                 ELSE y := . . .;
        .
        .
     END;
  BEGIN   (* main action statements *)
     .
     .
     readln(n);
     .
     .
     proc2(2*a+1,b);
     .
     .
     proc2(abs(a-10),b);
     .
     .
  END.
```

(*h*)

```
  PROGRAM sample(input,output);
  VAR a,b,c : char;
      t1,t2,t3 : boolean;

     FUNCTION upper(x : char) : boolean;
     BEGIN
        IF ord(x) <= 90 THEN upper := true
                        ELSE upper := false
     END;
  BEGIN   (* main action statements *)
     .
     .
     t1 := upper(a);
     .
     .
     t2 := upper(b,c);
     .
     .
     t3 := upper(a+b)
  END.
```

(*i*)

```
  PROGRAM sample(input,output);
  VAR a,b : integer;

     FUNCTION smaller : integer;
     BEGIN
        IF a < b THEN smaller := a
                 ELSE smaller := b
     END;
  BEGIN   (* main action statements *)
     .
     .
     writeln(' Smallest number: ',smaller);
     .
     .
  END.
```

(*j*)
```
PROGRAM sample(input,output);
VAR a,b : integer;

   PROCEDURE proc1(x,y : integer);
   VAR a,b : char;
   BEGIN
      .
      .
      .
   END;

BEGIN   (* main action statements *)
   .
   .
   proc1(a,b);
   .
   .
END.
```

(*k*)
```
PROGRAM sample(input,output);
VAR a,b,c : integer;
    x,y : real;

   FUNCTION funct1(x,y : integer) : real;
   VAR . . .;   (* local variables *)
   BEGIN
      .
      .
      c := . . .;
      .
      .
      funct1 := . . .
   END;

BEGIN   (* main action statements *)
   .
   .
   x := funct1(a,b);
   .
   .
   y := funct1(b,a-2);
   .
   .
END.
```

(*l*)
```
PROGRAM sample(input,output);
VAR x,y,z : real;

   FUNCTION smaller(FUNCTION f1(x : real) : real;
                    FUNCTION f2(x : real) : real) : real;
   BEGIN
      IF f1(x) < f2(x) THEN smaller := f1(x)
                       ELSE smaller := f2(x)
   END;
```

```
    BEGIN   (* main action statements *)
       .
       .
       y := smaller(sin,cos);
       .
       .
       z := smaller(log,sqrt);
       .
       .
    END.
```

(*m*)
```
    PROGRAM sample(input,output);
    VAR a,b,c : integer;

       FUNCTION functl(x,y : integer) : integer;
       VAR . . .;   (* local variables *)
       BEGIN
          .
          .
          .
          functl := . . .
       END;

       PROCEDURE procl(a,b,c : integer);
       VAR r,s,t : integer;
       BEGIN
          .
          .
          r := functl(s,t);
          .
          .
       END;

    BEGIN   (* main action statements *)
       .
       .
       c := functl(3,-1);
       .
       .
       procl(a,b+c,0);
       .
       .
       procl(a+b,functl(0,1),c+3);
       .
       .
    END.
```

(*n*)
```
    PROGRAM sample(input,output);
    VAR a,b,c : integer;

       PROCEDURE procl(a,b,c : integer);
       VAR r,s,t : integer;
       BEGIN
          .
          .
          r := functl(s,t);
          .
          .
       END;
```

```
   FUNCTION funct1(x,y : integer) : integer;
   VAR . . .;   (* local variables *)
   BEGIN
      .
      .
      .
      funct1 := . . .
   END;
BEGIN   (* main action statements *)
   .
   .
   c := funct1(3,-1);
   .
   .
   proc1(a,b+c,0);
   .
   .
   proc1(a+b,funct1(0,1),c+3);
   .
   .
END.
```

(o)

```
PROGRAM sample(input,output);
VAR a,b,c : integer;

   PROCEDURE test(VAR : x,y,z : integer);
   VAR r,s,t : integer;

      FUNCTION value(p,q : integer) : integer;
      VAR . . .;   (* local variables *)
      BEGIN
         .
         .
         .
         value := . . .
      END;
   BEGIN   (* test action statements *)
      .
      .
      r := value(s,t);
      .
      .
   END;
BEGIN   (* main action statements *)
   .
   .
   test(a,b,c);
   .
   .
   c := value(a,b);
   .
   .
END.
```

(p)
```
FUNCTION factorial(n : integer) : integer;
BEGIN
   factorial := n*factorial(n-1)
END;
```

(٣٦) عبر عن كل من الصيغ التالية بصيغة الاعادة الذاتية :

(a) $y = (x_1 + x_2 + \cdots + x_n)$
(b) $y = 1 - x + x^2/2 - x^3/6 + x^4/24 + \cdots + (-1)^n x^n/n!$
(c) $p = (f_1 * f_2 * \cdots * f_i)$

## مشاكل مبرمجة : Programming Problems

(٣٧) اكتب إجراء يحسب الجذور الحقيقية للمعادلة :

$$ax^2 + bx + c = 0$$

مستخدما العلاقة :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

افرض أن c , b , a هى مؤشرات قيمتها معطاه ، وأن $x_2 , x_1$ هى الجذور المحسوبة . افرض أيضا أن $b^2 > 4ac$ بحيث إن الجذور المحسوبة تكون دائما حقيقية ومميزة .

(٣٨) اكتب برنامجا كاملا بلغة البسكال يحسب الجذور الحقيقية لمعادلة .

$$ax^2 + bx + c = 0$$

مستخدما صيغة حساب الجذر المعطاه فى المشكلة رقم ٧ -٣٧ . أقرأ المعاملات c , b , a داخل الكمبيوتر فى المجموعة الرئيسية للبرنامج . ثم اتصل بالإجراء المكتوب فى المشكلة السابقة للحصول على الحل المطلوب . اكتب فى النهاية قيم المعاملات ، يتبعها قيم $x_2 , x_1$ المحسوبة . تاكد أن كل المخرجات لها عناوين واضحة .

| | | |
|---|---|---|
| a = 2 | b = 6 | c = 1 |
| 3 | 3 | 0 |
| 1 | 3 | 1 |

(٣٩) عدل الإجراء المكتوب فى المشكلة ٣٧ - ٧ ، بحيث إن كل جذور المعادلة .

$$ax^2 + bx + c = 0$$

تحسب مع معرفة c , b , a . لاحظ أن الجذور سوف تتكرر ( أى أنه سيكون هناك جذر حقيقى واحد فقط ) إذا ماكانت $b^2 = 4ac$ ، كما أن الجذور تكون مركبة أيضا إذا ماكان $b^2 < ac$ ، فى هذه الحالة يتحدد الجزء الحقيقى للكسر بأنه :

$$-b/2a$$

ويتحدد الجزء التخيلى للكسر بأنه :

$$(\pm\sqrt{4ac - b^2})i$$

حيث i تمثل $\sqrt{-1}$

(٤٠) عدل برنامج البسكال المكتوب فى المشكلة رقم ٧ -٣٩ . تاكد أن كل المخرجات لها عناوين واضحة . اختبر البرنامج مستخدما البيانات التالية .

| | | |
|---|---|---|
| a = 2 | b = 6 | c = 1 |
| 3 | 3 | 0 |
| 1 | 3 | 1 |
| 0 | 12 | −3 |
| 3 | 6 | 3 |
| 2 | −4 | 3 |

(٤١) اكتب دالة تسمح برفع عدد حقيقى أو صحيح إلى قوة صحيحة . وفى كلمات أخرى ... نحن نرغب فى تقويم الصيغة :

$$y = x^n$$

حيث x , y متغيران من النوع الصحيح أو الحقيقى ، و n متغير من النوع الصحيح . ( لاحظ أن x , y لابد أن يكونا من نفس النوع دائما )

(٤٢) اكتب برنامجا كاملا بلغة البسكال يقرأ قيما عددية لكل من x , n لتقويم الصيغة الموجودة فى المشكلة رقم ٤١ - ٧ .

$$y = x^n$$

مستخدما الدالة المكتوبة فى المشكلة رقم ٧ -٤١ ، ثم اخرج النتائج المحسوبة . اختبر البرنامج مستخدما البيانات التالية :

| x | n |
|---|---|
| 2 | 3 |
| 2 | 12 |
| 2 | −5 |
| −3 | 3 |
| −3 | 7 |
| −3 | −5 |
| 1.5 | 3 |
| 1.5 | 10 |
| 1.5 | −5 |
| 0.2 | 3 |
| 0.2 | 5 |
| 0.2 | −5 |

(٤٣) وسع الدالة المكتوبة فى المشكلة رقم ٧ -٤١ بحيث يمكن رفع x لأى قوة ، سـواء أكانـت حقيقية أم صـحيحـة ( ملاحظة : استخدم اللوغاريتم إذا كان الأس من النوع الحقيقى . الصيغة هى $y = x^n = e^{(n \ln x)}$ تذكر أن تختبر القيم السالبة لـ x ) . حدد هذه الدالة فى البرنامج المكتوب فى المشكلة رقم ٧ - ٤٢ . اختبر البرنامج مستخدما البيانات المعطاه فى المشكلة رقم ٧ - ٤٢ مع البيانات الإضافية التالية .

| x | n |
|---|---|
| 2 | 0.2 |
| 2 | −0.8 |
| −3 | 0.2 |
| −3 | −0.8 |
| 1.5 | 0.2 |
| 1.5 | −0.8 |
| 0.2 | 0.2 |
| 0.2 | −0.8 |
| 0.2 | 0.0 |

(٤٤) أعد برنامج شفرة الرموز المعطى فى مثال ٦ -٣١ بحيث إنه يستخدم إجراء . ترجم ونفذ البرنامج للتأكد من أنه يعمل بطريقة صحيحة .

(٤٥) عدل البرنامج لحساب حل معادلة جبرية معطاه فى مثال ٦ -١٤ بحيث إن كل تكرار يحدث داخل دالة . ترجم ونفذ البرنامج للتأكد من أنه يعمل بطريقة صحيحة .

(٤٦) عدل برنامج حساب متوسط قائمة بالأعداد المعطاه فى مثال ٧ – ٨ ، بحيث إنه يستخدم دالة . اختبر البرنامج مستخدما العشرة أعداد التالية :

| | |
|---|---|
| 27.5 | 87.0 |
| 13.4 | 39.9 |
| 53.8 | 47.7 |
| 29.2 | 8.1 |
| 74.5 | 63.2 |

(٤٧) يمكن أن يعدل البرنامج المعطى فى مثال ٧ – ١٤ببساطة لتقليل ( لتدنية ) minimize دالة فى x . مثل هذه العملية للتقليل ( للتدنية ) يمكن أن توفر لنا أسلوب مرتفع الكفاءة لحساب جذور معادلة جبرية غير خطية . افرض مثلا أننا نريد إيجاد قيمة خاصة لـ x تتسبب فى أن الدالة f (x) = 0 . الدالة التقليدية من هذا النوع يمكن أن تأخذ الصورة .

$$f(x) = x + \cos(x) - 1 - \sin(x).$$

إذا جعلنا $g(x) = f(x)^2$ ، فتكون الدالة g (x) موجبة دائما ، إلا إذا كانت قيم x هى جذور الدالة المعطاه ، أى للدالة f (x) ، وبالتالى g (x) ، والتى تساوى صفرا . وعلى هذا ... فأى قيمة لـ x تدنى g (x) تكون جذرا للمعادلة f (x) = 0 أيضا . عدل البرنامج الموضح فى مثال ٧ – ١٤لتدنية دالة معطاه . استخدم البرنامج فى الحصول على جذور المعادلات التالية :

$$(a)\quad x + \cos(x) = 1 + \sin(x), \qquad \pi/2 < x < \pi$$
$$(b)\quad x^5 + 3x^2 = 10, \qquad 0 <= x <= 3$$

(٤٨) عدل البرنامج الموضح فى مثال ٧ – ١٨، بحيث يحاكى تسلسل المباراة تلقائيا بطريقة غير متكررة . حدد أعداداً من النوع الصحيح يحدد إجمالى عدد مرات المكسب ومتغير مدخلات صحيح قيمته تحدد عدد المباريات التى تحاكى .

استخدم البرنامج فى محاكاة عدد كبير من المباريات ( أى 1000 مباراة مثلا ) . قدر احتمال المكسب عند لعب المباراة . ( هذه القيمة 6 معبر عنها ككسر عشرى 6 تساوى عدد مرات المكسب مقسوما على إجمالى عدد المباريات الملعوبة . إذا تعدى الاحتمال 0.50 فهذا فى صالح اللاعب ) .

(٤٩) أعد كتابة كل من البرامج التالية ، بحيث إنها تشمل إجراء واحد على الأقل ، أو دالة واحدة على الأقل . كن صريحا عند اختيارك المؤشرات .

(أ) حساب المتوسط المرجح لقائمة أعداد ( انظر المشكلة رقم ٥٠ الجزء a من الفصل السادس ) .

(ب) حساب الضرب التراكمى لقائمة أعداد ( انظر المشكلة رقم ٥٠ الجزء b من الفصل السادس ) .

(ج) حساب المتوسط الهندسى لقائمة أعداد ( انظر المشكلة رقم ٥٠ الجزء c من الفصل السادس ) .

(د) حساب وجدولة قائمة بالاعداد الأولية ( انظر المشكلة رقم ٥٠ الجزء f من الفصل السادس ) .

(هـ) حساب جيب الزاوية مستخدما الطريقة المذكورة ( انظر المشكلة رقم ٥٠ الجزء i من الفصل السادس ) .

(و) حساب أقساط إعادة دفع القرض ( انظر المشكلة رقم ٥٠ الجزء j من الفصل السادس ) .

(ز) تحديد متوسط درجات الامتحان لكل طالب في الفصل كما هو مذكور فى المشكلة رقم ٦ – ٥٠ الجزء k .

(٥٠) اكتب برنامجا كاملا بلغة البسكال لحل كل مشكلة من المشاكل المذكورة أدناه استخدم إجراءات ودوال كلما كان ذلك ممكنا . ترجم ونفذ كل برنامج باستخدام البيانات المعطاه مع وصف المشكلة .

(أ) افرض أنك تضع مبلغا معينا من المال A فى حساب توفير مع بداية كل سنة لعدد n من السنين . فإذا كان معدل الفائدة i % سنويا ، فإن المبلغ المتراكم بعد انقضاء n سنة هو F ، حيث :

$$F = A\ [(1+i/100) + (1+i/100)^2 + (1+i/100)^3 + \cdots + (1+i/100)^n]$$

اكتب برنامج بسكال على هيئة حوار لتحديد مايلى :

١) المبلغ المتراكم بعد 30 سنة إذا كان المبلغ المودع فى بداية كل سنة هو 1000 دولار ، ومعدل الفائدة 6% فى السنة .

٢) المبلغ الذى يجب إيداعه فى بداية كل سنة لكى يصبح المبلغ المتراكم 100,000 دولار بعد انقضاء 30 سنة بافتراض أن معدل الفائدة 6% سنويا .

حدد فى كل حالة المبلغ غير المعروف أولا ، ثم اعمل جدولاً يوضح إجمالى المبلغ الذى يتراكم مع نهاية كل سنة . استخدم الدالة المكتوبة فى المشكلة رقم ٧ –١٤ لأداء العمليات الأسية .

(ب) عدل البرنامج السابق لحساب الفائدة المركبة كل ربع سنة ، بدلا من حسابها كل سنة . قارن النتائج التى تحصل عليها فى كل من الحالتين . ملاحظة : الصيغة المناسبة هى :

$$F = A\ [(1 + i/100m)^m + (1 + i/100m)^{2m} + (1 + i/100m)^{3m} + \cdots + (1+i/100m)^{nm}]$$

حيث m تمثل عدد الفترات التى يدفع عنها فوائد فى السنة .

(ج) تتحدد تكاليف قرض البناء بطريقة تجعل المقترض يدفع نفس المبلغ ( قسط ثابت ) للمؤسسة الدائنة شهريا طوال فترة سداد القرض . ويتغير جزء القسط الشهرى المطلوب كفائدة على الموازنة القائمة على أية حال من شهر لشهر . ففى بداية فترة سداد القرض يكون معظم القسط ممثلا لجزء الفائدة ، وجزء بسيط يمثل مايسدد من صلب القرض . وبالتدريج تصبح الموازنة القائمة أقل ، مما يقلل من الفائدة الشهرية ، وزيادة الجزء المسدد من صلب القرض . وعلى هذا ... تقل موازنة القرض بمعدل متزايد .

ويعرف الذين يشترون المنازل المبلغ الذين فى حاجة لاقتراضه ، والوقت اللازم لإعادة دفعه . بعد ذلك ... فإنهم يطلبون القروض من مؤسسات الإقراض ، بعد معرفتهم القسط الشهرى وقيمة الفائدة . كما يجب أن يهتموا أيضا بقيمة الفائدة المسددة شهريا ، وقيمة الجزء المسدد شهريا من صلب القرض ، ومايتبقى عليهم ، وذلك فى كل شهر .

اكتب برنامجا بلغة البسكال يمكن أن تستخدمه مؤسسات الإقراض لتوفير هذه المعلومات للعملاء . أفرض أن قيمة القرض ومعدل الفائدة السنوى وفترة سداد القرض محددة . القسط الشهرى يحسب كما يلى :

$$A = \frac{iP\,(1+i)^n}{(1+i)^n - 1}$$

حيث : A = القسط الشهرى بالدولار .

P = إجمالى قيمة القرض بالدولار .

i = معدل الفائدة الشهرى ككسر عشرى

( أى أن 1/2% تكتب 0.005 ) .

n = إجمالى عدد الأقساط الشهرية .

ويمكن حساب مايدفع كفائدة شهرية من العلاقة التالية :

$$I = iB$$

حيث I = مايدفع كفائدة شهرية بالدولار .

B = الموازنة القائمة الحالية بالدولار .

الموازنة القائمة الحالية تساوى المبلغ الأصلى للقرض ، مطروحا منه مجموع المدفوعات السابقة ناحية الأساس . مايدفع شهريا ناحية الأساس ( أى المبلغ المستخدم لتقليل الموزانة القائمة ، أو لتقليل صلب القرض ) هو :

$$T = A - I$$

حيث T = مايدفع شهريا ناحية الأساس .

استخدم الدالة المكتوبة فى المشكلة رقم ٧ -٤١لأداء العمليات الأسية .

استخدم البرنامج فى حساب تكلفة قرض 50,000 دولار ، يسدد على 25 سنة ، بمعدل فائدة سنوية % 8 . ثم أعد الحسابات بمعدل فائدة % 8.5 . ماهى درجة معنوية % 0.5 فى معدل الفائدة عبر فترة سداد القرض ؟

(د) الطريقة المستخدمة فى حساب تكلفة قرض المنزل فى المشكلة السابقة تعرف بأنها طريقة القسط الثابت con-stant payment method ، حيث إن كل الأقساط الشهرية متساوية . افرض أنه بدلا من ذلك يحسب القسط الشهرى باستخدام طريقة الفائدة البسيطة . أى افرض أن نفس الكمية تطبق ناحية تقليل مقدار القرض كل شهر ، أى

$$T = P/n$$

على أية حال تعتمد الفائدة الشهرية على قيمة الموزانة القائمة ، أى أن

$$I = iB$$

وعلى هذا ... فإن إجمالي مايدفع شهريا A هو

$$A = T + I$$

سوف يقل في كل شهر حتى ينتهي القرض .

اكتب برنامجا بلغة البسكال لحساب تكلفة قرض مستخدما هذه الطريقة في إعادة الدفع . اكتب عناوين واضحة للمخرجات . استخدم البرنامج في حساب تكلفة قرض قيمته 50,000 دولار ، وفترة سداده 25 سنة ، ومعدل الفائدة 8 % سنويا . قارن النتائج بالنتائج التي تحصل عليها من المشكلة رقم ٧ -٥٠ (ج) .

(هـ) افرض أننا لدينا عدد من النقاط الوثابة discete $(x_1, y_1)$ $(x_2, y_2)$ و . . . $(x_n, y_n)$ ، والتي تقرأ من منحنى $y = f(x)$ ، حيث x محاطة بين $x_1$ ، $x_n$ ونريد أن نقرب المساحة تحت المنحنى ، وذلك بتجزئة المنحنى إلى عدد من المستطيلات الصغيرة ، وحساب مساحات هذه المستطيلات ( هذا مايعرف بقاعدة شبه المنحرف trapezoidalrule ) . والصيغة المناسبة هي :

$$A = (y_1 + y_2)(x_2 - x_1)/2 + (y_2 + y_3)(x_3 - x_2)/2 + \cdots + (y_{n-1} + y_n)(x_n - x_{n-1})/2$$

لاحظ أن متوسط ارتفاع كل مستطيل يعطى بالعلاقة

$$(y_i + y_{i+1})/2$$

وأن عرض المستطيل يساوى

$$(x_{i+1} - x_i);\ i = 1, 2, \ldots, (n - 1)$$

حيث :

اكتب برنامجا بلغة البسكال لتنفيذ هذه العملية ، مستخدما دالة في تقويم الصيغة الرياضية $y = f(x)$ ، استخدم البرنامج في حساب المساحة تحت المنحنى $y = x^3$ بين الحدود x = 1 , x = 4 . حل هذه المشكلة أولا مستخدما 16 نقطة على أبعاد متساوية من بعضها ، ثم مستخدما 61 نقطة ، وبعد ذلك مستخدما 301 نقطة . لاحظ أن دقة الحل تتحسن مع زيادة عدد النقاط . ( الإجابة الدقيقة لهذه المشكلة هي 63.75 ) .

(و) تصف المشكلة السابقة قاعدة شبه المنحرف trapezoidal rule لحساب المساحة تحت المنحنى $y(x)$ ، حيث تستخدم مجموعة من القيم المجدولة $(x_1, y_1)$ $(x_2, y_2)$ , . . . , $(x_n, y_n)$ في وصف المنحنى . إذا كانت القيم المجدولة لـ x على مسافات متساوية بعضها البعض ، فإن المعادلة المعطاه في المشكلة السابقة يمكن تبسيطها لتاخذ الشكل التالي :

$$A = (y_1 + 2y_2 + 2y_3 + 2y_4 + \cdots + 2y_{n-1} + y_n)h/2$$

حيث h هي المسافة بين قيمتين متتاليتين من قيم x .

وهناك طريقة تستخدم عندما يكون هناك عدد زوجي من المناطق المتساوية ، أي عدد فردي من نقاط البيانات ، وهي قاعدة سيمبسون Simpson's rule . ومعادلة الحسابات المستخدمة مع قاعدة سيمبسون هي :

$$A = (y_1 + 4y_2 + 2y_3 + 4y_4 + 2y_5 + \cdots + 4y_{n-1} + y_n)h/3$$

إحدى قيم h المعرفة ، فإن هذه الطريقة تعطى نتيجة أكثر دقة من طريقة قاعدة شبه المنحرف ( لاحظ أن الطريقة تحتاج لنفس كمية الحسابات التى تحتاجها طريقة شبه المنحرف تقريبا ) .

اكتب برنامجا بلغة البسكال لحساب المساحة تحت المنحنى ، مستخدما أى طريقة من الطرق سالفة الذكر ، مفترضا أن هناك عددًا فرديًا لنقاط بيانات متساوية المسافة بينها . نفذ كل طريقة بإجراء منفصل ، مستخدما دالة فى تقويم y (x) .

استخدم البرنامج فى حساب المساحة تحت المنحنى الممثل بالمعادلة .

$$y = e^{-x^2}$$

حيث x تتراوح من 0 إلى 1 . احسب المساحة مستخدما كل طريقة ، ثم قارن النتائج بالنتيجة الصحيحة ، وهى :

$$A = 0.7468241$$

(ز) مازال هناك طريقة أخرى لحساب المساحة تحت المنحنى تستخدم أسلوب مونت كارلو Monte Carlo ، والتى تستخدم إنتاج أعداد عشوائية . افرض أن المنحنى y = f (x) موجب لأى قيمة من قيم x التى تقع بين حد أدنى محدد x = a ، وحد أعلى محدد x = b . دع أكبر قيمة لـ y داخل هذه الحدود هى *y . وتنفذ طريقة مونت كارلو كما يلى :

١) ابدأ بعداد قيمته تساوى صفرًا .

٢) انتج عددًا عشوائيًا $r_x$ ، تقع قيمته بين b , a .

٣) قوّم $y(r_x)$

٤) انتج عددًا عشوائيًا $r_y$ ، تقع قيمته بين 0 , *y .

٥) قارن $r_x , r_y$ . فإذا كان $r_y$ أقل من أو يساوى $r_x$ ؛ فإن هذه النقطة تقع على المنحنى أو تحته . وعند ذلك تزداد قيمة العداد بمقدار 1 .

٦) أعد الخطوات من 2 إلى 5 عددا كبيرا من المرات . وكل مرة تسمى دورة cycle .

٧) وعند الانتهاء من إتمام عدد معين من الدورات ، فإن كسر النقاط الذى يقع على المنحنى أو تحته . والمسمى F يحسب بأنه قيمة العداد مقسوما على إجمالى عدد الدورات . ويتم الحصول على ذلك على المساحة تحت المنحنى كمايلى :

$$A = Fy^*(b - a)$$

اكتب برنامجا بلغة البسكال لتنفيذ هذه العمليات . استخدم هذا البرنامج فى إيجاد المساحة تحت المنحنى $y = e^{-x^2}$ التى تقع بين الحدين a = 0 , b = 1 . حدد عدد الدورات اللازمة للحصول على إجابة دقيقة لأقرب ثلاثة أرقام معنوية . قارن وقت الكمبيوتر اللازم لهذه المشكلة مع الوقت اللازم للمشكلة السابقة . أى طريقة أفضل ؟

(ح) يمكن إنتاج التوزيع الطبيعى للتغير العشوائى x والذى له متوسط μ وانحراف معيارى σ ، وذلك باستخدام الصيغة التالية :

$$x = \mu + \sigma \frac{\sum_{i=1}^{N} r_i - N/2}{\sqrt{N/12}}$$

حيث $r_i$ هو عدد عشوائى له توزيع طبيعى ، وتقع قيمته بين 0 , 1 . وغالبا مايتكرر اختيار قيمة 12 عند استخدام هذه الصيغة . وأساس هذه الصيغة هو نظرية الحد المركزى central limit theorem ، والتى تحدد قيما للمتوسط لتغيرات عشوائية ذات توزيع طبيعى .

اكتب برنامجا بلغة البسكال ينتج عدداً محدداً من التغيرات العشوائية الموزعة توزيعا طبيعيا ، لها متوسط معروف ، وانحراف معيارى معروف . اجعل عدد التغيرات العشوائية والمتوسط والانحراف المعيارى مؤشرات مدخلات .

(ط) اكتب برنامجا بلغة البسكال يسمح لشخص أن يلعب لعبة tic - tac - toe مع الكمبيوتر . اكتب البرنامج بحيث يمكن للكمبيوتر أن يكون اللاعب الأول أو اللاعب الثانى . فإذا ماكان الكمبيوتر هو اللاعب الأول ، دع أول حركة يتم إنتاجها عشوائيا . اكتب الحالة الكاملة للمباراة بعد كل حركة . دع الكمبيوتر يحدد من الذى يكسب عند حدوث ذلك .

(٥١) اكتب برنامجا كاملا بلغة البسكال لكل مشكلة من المشاكل التالية ، على أن يحتوى على دالة إعادة ذاتية .

(أ) تحديد قيمة عدد فابوناكى $F_n$ رقم n ، حيث :

$$F_n = F_{n-1} + F_{n-2} \text{ and } F_1 = F_2 = 1$$

( انظر المشكلة رقم ٥٠ - ٦ الجزء هـ ) . دع قيمة n كمؤشر مدخلات ، وهى أكبر من 2 .

(ب) يمكن حساب Legendre polynomials بواسطة الصيغ $P_0 = 1, \ P_1 = x,$

$$P_n = \frac{2n-1}{n} x P_{n-1} - \frac{n-1}{n} P_{n-2}$$

حيث n = 2 , 3 , 4 , ... , x أى عدد حقيقى يقع بين 1- , 1 ( لاحظ أن Legendre polynomials هى كميات من النوع الحقيقى ) دع قيم x , n مؤشرات مدخلات .

## الفصل الثامن

# البيانات البسيطة التى يعرفها المستفيد

# User-Defined Simple-Type Data

سبق أن رأينا بالفعل أن هناك نوعين من البيانات البسيطة العامة فى البسكال ، وهما : بيانات قياسية Standard - type data وبيانات يعرفها المستفيد user - defined data ( انظر القسم رقم ٢ من الفصل السابع ) . ويمكن تجزئة هذين النوعين إلى مايلى :

١ - بيانات قياسية :

أ - صحيحة

ب - حقيقية

ج - حرفية .

د - بوليان .

٢ - بيانات يعرفها المستفيد :

أ - نوع متعدد enumerated

ب - نوع مدى جزئى subrange

وقد سبق مناقشة استخدام البيانات القياسية بالتفصيل فى الفصل الثالث من الكتاب . ونركز انتباهنا الآن إلى البيانات البسيطة التى يعرفها المستفيد . وسوف نعتبر بصفة خاصة البيانات المتعددة ، والبيانات من نوع المدى الجزئى . وسوف يتضح لنا أنه يمكن توضيح البيانات البسيطة ، وكيفية استخدامها بكفاءة فى برنامج البسكال .

## ١ - بيانات من النوع المتعدد : 1. ENUMERATED - TYPE DATA

تحتوى البيانات المتعددة على ترتيب تسلسل من المعرفات ، حيث يفسر كل معرف كعنصر بيانات مستقل يصاحب عناصر البيانات هذه مع بعضها اسم خاص يقوم بتعريف نوع البيانات . ومصاحبة اسم نوع البيانات مع عناصر البيانات الفردية يتحدد فى تعريف النوع type . ويكتب تعريف النوع بصفة عامة كمايلى :

```
TYPE name = (data item 1,data item 2, . . .,data item n)
```

حيث name هو اسم بيانات من النوع المتعدد ، و data item 1 و data item 2 هى أسماء البيانات الفعلية .

### مثـال (٨-١)

نعرف فى هذا المثال عنصر البيانات من النوع المتعدد ، اسمه days . وعناصر البيانات الفعلية تصبح أيـام

الأسبوع ، وهى :

sunday, monday, tuesday, wednesday, thursday,

friday and saturday

وعلى هذا ... فتعريف النوع يكتب كمايلى :

```
TYPE days = (sunday,monday,tuesday,wednesday,thursday,friday,saturday);
```

لاحظ أن عناصر البيانات مكتوبة بين قوسين ومفصولة عن بعضها بفواصل .

ويجب أن يكون مفهوما أن هذه هى القيم الوحيدة التى يمكنها أن تصاحب نوع days .

حيث إن البيانات من النوع المتعدد لها تعريف بترتيب متسلسل ، فيمكن استخدام المؤثرات العلاقية معها فى تكوين تعبيرات بوليان . كما يمكن استخدام الدالتين القياسيتين succ , pred لتحديد أى عناصر البيانات تسبق عنصر بيانات محدد ، وأيها يتبعه .

## مثـال (٨-٢)

اعتبر days ، وهو من بيانات النوع المتعدد الذى سبق تعريفه فى المثال السابق . وفيما يلى العديد من تعبيرات بوليان التى تستخدم عناصر البيانات وقيمها المناظرة .

| التعبير | القيمة |
|---|---|
| sunday < tuesday | صحيح |
| wednesday >= saturday | خطأ |
| monday <> friday | صحيح |
| pred(friday) = thursday | صحيح |
| succ(friday) = saturday | صحيح |
| succ(tuesday) <> pred(thursday) | خطأ |

ويمكن استخدام الدالة القياسية ord أيضا مع البيانات من النوع المتعدد . ولعمل ذلك يجب أن يكون مفهوما أن أول عنصر بيانات يحدد له الرقم 0 ، والثانى يحدد له الرقم 1 وهكذا . وعلى هذا ... فإذا كان هناك n عنصر بيانات داخل بيانات من النوع المتعدد ، فإن رقم الترتيب يتراوح من 0 إلى n-1 .

## مـثال (٨-٣)

اعتبر مرة أخرى days كبيانات النوع المتعدد ، والمعرفة فى مثال (٨-١) . العلاقات التالية صحيحة :

```
succ(sunday) = monday        ord(sunday) = 0
pred(monday) = sunday        ord(monday) = 1
succ(monday) = tuesday       ord(tuesday) = 2
```

```
pred(friday) = thursday          ord(saturday) = 6
succ(friday) = saturday
pred(saturday) = friday
```

## ٢ – بيانات من نوع المدى الجزئى : 2. SUBRANGE - TYPE DATA

يشير المدى الجزئى subrange إلى جزء من مدى أصلى لنوع بيانات بسيط ومرتب . وبيانات المدى الجزئى هى عناصر بيانات تقع داخل هذا المدى الجزئى ، مكونه فئة جزئية من البيانات المرتبة والمتماسة contiguous . ويشار الى نوع البيانات الأصلى بأنه بيانات من النوع المضيف host . وكل عنصر بيانات من العناصر التى تقع داخل المدى الجزئى يعتبر من النوع المضيف .

ويمكن تطبيق مفهوم المدى الجزئى على أى مجموعة بيانات بسيطة مرتبة . ويشمل ذلك البيانات من النوع المتعدد التى سبق تعريفها ، كما يشمل ثلاثة أنواع من البيانات القياسية ، وهى البيانات الصحيحة والحرفية والبوليان . ( لاحظ أن البيانات الحقيقية لايمكن cannot استخدامها فى تعريف مدى جزئى ) .

والصيغة العامة لتعريف المدى الجزئى تكتب كما يلى :

```
TYPE name = first data item..last data item
```

حيث name هو اسم نوع بيانات المدى الجزئى . كما أن first data item هو أول عنصر بيانات فى الترتيب داخل المدى الجزئى ، و last data item هو آخر عنصر بيانات فى الترتيب داخل المدى الجزئى . ويحتوى المدى الجزئى الكامل على كل عناصر البيانات الموجودة بين هذين الحدين ( بما فيها الحدين أيضا ) . لاحظ أنه يجب أن يفصل أول عنصر بيانات عن آخر عنصر بيانات بوجود نقطتين متتاليتين .

### مثال (٨-٤)

فيما يلى عدة بيانات من النوع الذى يعرفه المستفيد .

```
TYPE days = (sunday,monday,tuesday,wednesday,thursday,friday,saturday);
     weekdays = monday..friday;
     month = 1..31;
     caps = 'A'..'Z';
```

يعيد أول سطر تعريف النوع المتعدد المعطى فى مثال (٨ – ١) . ونعرف فى السطر الثانى نوع بيانات من المدى الجزئى يسمى weekdays يحتوى على فئة جزئية من الأيام ( وهى بصفة خاصة عناصر البيانات الخمسة friday , thursday , wednesday , tuesday , monday ) . لاحظ أن تعريف النوع المتعدد days يجب أن يسبق تعريف نوع المدى الجزئى weekdays .

ويعرف السطر الثالث نوع مدى جزئى آخر يسمى month . ويحتوى هذا النوع من البيانات على أرقام متتالية من 1 إلى 31 . وعلى هذا ... فإن نوع البيانات يكون فئة جزئية من النوع القياسى للبيانات الصحيحة .

أخيرا يعرف السطر الأخير نوع البيانات من المدى الجزئى ، وهو Caps الذى يحتوى على الحروف العلوية ( من A إلى Z ) . لاحظ أن آخر نوع من البيانات هذا هو فئة جزئية من البيانات الحرفية القياسية .

وحيث إن البيانات من نوع المدى الجزئى دائما ماتكون مرتبة ، فيمكن استخدام عناصر البيانات الفردية مع المؤثرات العلاقية لتكوين تعبيرات بوليان . كما يمكن استخدامها أيضا مع الدوال القياسية ord , succ , pred بنفس الطريقة التى تستخدم بها هذه الدوال مع البيانات البسيطة المرتبة الأخرى . وسوف نرى بعض أمثلة توضح استخدام بيانات من نوع المدى الجزئى فيما بعد فى هذا الفصل .

## ٣ – المزيد عن التوضيحات : 3. MORE ABOUT DECLARATIONS

دعنا نستعرض مرة أخرى الهيكل الشامل لبرنامج البسكال . دعنا نفحص بصفة خاصة جزء التوضيحات من البرنامج ، حيث إننا قادرون الآن على فهم معنى كل العناصر التى يحتويها . ( تذكر أن هذه المادة سبق تقديمها فى القسم الخامس من الفصل الأول وفى القسم الثانى من الفصل الخامس .

فيما يلى تخطيطا شاملا للبرنامج :

١ – عنوان .

٢ – صلب البرنامج

أ – توضيحات :

١ – عناوين .

٢ – ثوابت

٣ – تعريفات الأنواع .

٤ – متغيرات .

٥ – إجراءات ودوال .

ب – عبارات .

ويبدأ قسم التوضيحات بعناوين العبارات ، والتى تستخدم مع عبارات GOTO ( انظر القسم الثامن من الفصل السادس ) . ويتبع ذلك تعريفات الثوابت ( القسم الثامن من الفصل الثانى ) ، ثم تعريفات الأنواع ( القسمين الأول والثانى من الفصل الثامن ) . ثم يلى ذلك توضيحات المتغيرات ( القسم التاسع من الفصل الثانى ) ، يتبعها توضيحات الإجراءات والدوال ( القسمين الأول والرابع من الفصل السابع ) .

وحقيقية أن تعريفات النوع تسبق توضيحات المتغيرات لها معنى خاص . فهذا يسمح بتوضيح المتغيرات بالنسبة لأنواع البيانات التى يعرفها المستفيد ، وبالنسبة أيضا لأنواع البيانات القياسية . ومقدرتنا على تمييز أنواع البيانات التى يعرفها المستفيد لها على ذلك أهمية كبيرة .

ومن الممكن أيضا دمج تعريف نوع المدى الجزئى المستخدم مع توضيح المتغير المناظر . وهذا مريح بصفة خاصة إذا ما استخدم نوع المدى الجزئى توضيحا واحدا فقط لمتغير .

وتوضيح المتغيرات التى تكون بياناتها من النوع الذى يحدده المستفيد موضح فى المثال التالى .

## مـثال (٨-٥)

اعتبر تعريفات النوع وتوضيحات المتغيرات التالية : ( أعد إنتاج تعريفات النوع من مثال ٨ - ٤ ) .

```
TYPE days = (sunday,monday,tuesday,wednesday,thursday,friday,saturday);
     weekdays = monday..friday;
     month = 1..31;
     caps = 'A'..'Z';
VAR workdays,holidays : weekdays;
    dayofmonth : month;
    hoursworked : 1..24;
    grosspay,netpay : real;
    employeenumber : 1..maxint;
```

لاحظ أن المتغيرات weekdays , holidays هى من نوع weekdays . وعلى هذا ... فكل من هذه المتغيرات يمكن أن يأخذ قيمة monday , tuesday , wednesday , thursday , friday . وبالمثل المتغير dayofmonth هو من نوع month . وعلى هذا ... يمكن أن يأخذ أى قيمة صحيحة تتراوح من 1 إلى 31 .

أعتبر الآن المتغير hoursworked . هذا متغير من نوع المدى الجزئى ، ويشبه dayofmonth الذى يمكن أن يأخذ أى قيمة صحيحة تتراوح من 1 إلى 24 . لاحظ على أية حال أن تعريف النوع وتوضيح المتغير مندمجان معا فى هذه الحالة . وقد كان من الممكن بالطبع استبدال هذا التوضيح الفردى بما يلى :

```
TYPE hour = 1..24;
VAR hoursworked : hour;
```

بالرغم من أن هذا متعب بعض الشئ .

لاحظ أنه بإمكاننا أيضا أن ندمج تعريف نوع month مع توضيح المتغير dayofmonth . وعلى هذا ... يمكننا كتابة مايلى :

```
VAR dayofmonth : 1..31;
```

وأخيرا فإن آخر توضيحين لمتغيرات يحددان أن grosspay , netpay متغيرات من النوع الحقيقى ، وأن employeenumber متغير موجب من النوع الصحيح . ( لاحظ أن employeenumber متغير من نوع المدى الجزئى فعلا ) وعلى هذا نرى توضيحات متغيرات تحتوى على أنواع بيانات قياسية ، وعلى أنواع بيانات يعرفها المستفيد أيضا فى هذا المثال .

## ٤ - استخدام البيانات التى يعرفها المستفيد : - 4. UTILIZING USER - DEFINED DATA

بعد تعريف البيانات من النوع المتعدد ونوع المدى الجزئى ، يمكن استخدام هذه البيانات ، مثل عناصر البيانات بسيطة النوع الأخرى داخل برنامج البسكال . وهذا يشمل استخدامها كمتغيرات تحكم control variables فى مكونات FOR وقائمين بالاختيار فى مكونات CASE .

## مـثال (٨-٦)

فيما يلى مثالاً لبيانات من النوع المتعدد مستخدمة كمتغيرات تحكم فى مكون FOR .

```
FOR weekdays := monday TO friday DO
   BEGIN
       .
       .
       .
   END;
```

متغير التحكم weekdays معرف فى مثال (٨-٥) . وعلى هذا ... فإن weekdays يأخذ قيمة monday أثناء أول مسار فى الدورة ، ثم يأخذ tuesday أثناء المسار الثانى للدورة وهكذا ، حتى يأخذ friday أثناء آخر ( خامس ) مسار .

## مـثال (٨-٧)

اعتبر الآن استخدام نفس المتغير المتعدد weekdays كقائم بالاختيار فى مكون CASE .

```
CASE weekdays OF
     monday    : writeln(' First workday');
     tuesday   : writeln(' Second workday');
     wednesday : writeln(' Third workday');
     thursday  : writeln(' Fourth workday');
     friday    : writeln(' Last workday')
END;
```

تكتب رسالة واحدة فقط فى كل مرة يقابل فيها مكون CASE طبقا للـقيمة التى تـسكون محـدودة للمتغير weekdays .

سبق أن رأينا أنه يمكن استخدام بيانات متعددة النوع ، ومن نوع المدى الجزئى فى إنتاج تعبيرات بوليان .

يمكن أيضا لبيانات المدى الجزئى من النوع الصحيح أن تظهر فى تعبيرات من النوع الصحيح ، وتتبع بالضرورة نفس القواعد التى تتبعها البيانات القياسية صحيحة النوع .

ويمكن أن تحدد للمتغيرات من النوع المتعدد ، ومن نوع المدى الجزئى قيم مناظرة لها من النوع المناسب . وعلى هذا ... يمكن تحديد أحد عناصر النوع المتعدد لمتغير متعدد من نفس النوع . وبالمثل يمكن تحديد عنصر من عناصر المدى الجزئى لمتغير مدى جزئى من نفس النوع .

## مثال (٨ – ٨) :

افرض أن holidays متغير متعدد ، وأن hoursworked , employeenumber متغيران مدى جزئى كما هى معرفة فى مثال (٨ – ٥) . وعلى هذا ... فإن عبارات التحديد التالية صحيحة :

```
employeenumber := 12345;
holidays := thursday;
hoursworked := 8;
```

لايمكن إدخال البيانات متعددة النوع داخل الكمبيوتر عن طريق عبارة read أو readln . وعلى أية حال ... يمكن إدخال البيانات من نوع المدى الجزئى صحيحة أو حرفية لتوفر قيم تقع داخل المدى الجزئى المسموح به . ونفس القيود موجودة على كتابة البيانات من الكمبيوتر عن طريق عبارة write أو writeln .

ويسمح استخدام البيانات من النوع المتعدد أن يعبر عن برنامج البسكال بالنسبة لمتغيرات مرتبطة ارتباطا وثيقا بتعريف المشكلة المناظر . وهذه العملية يوصى باستخدامها ، حيث إنها تزيد من الوضوح ومن بساطة المنطق . كما أن ذلك يسهل من تصحيح أخطاء البرنامج .

ومزايا استخدام بيانات المدى الجزئى شبيهه بذلك أيضا ، بالرغم من قلة تأثيرها . وبصفة خاصة فإن استخدام بيانات المدى الجزئى تبسط البرنامج وتحسن من قراءته ، كما يمكن أن ينتج عنها تقليل فى مساحة الذاكرة المطلوبة .

## مــثال (٨-٩)

عدد *الأيام بين تاريخين* . افرض أنه تم إدخال تاريخين مختلفين داخل الكمبيوتر ، وأننا نريد تحديد عدد الأيام الموجودة بين هذين التاريخين . إحدى طرق علاج هذه المشكلة هى تحديد عدد الأيام منذ كل من التاريخين ، وحتى تاريخ أساسي مشترك سابق لهما . ويمكن عند ذلك تحديد عدد الأيام الموجود بين هذين التاريخين بطرح الرقميــن من بعضــهما .

دعنا نختار أول يناير 1960 كتاريخ أساسى . إذا ماأدخل التاريخان فى صورة mm dd yy ( حيث mm تمثل رقما صحيحا يعبر عن الشهر ، dd رقما آخر صحيحا يعبر عن اليوم ، yy رقما ثالثا صحيحا يعبر عن السنة ) ، فإن عدد الأيام منذ تاريخ الأساسى حتى التاريخ المعطى يمكن تحديده بالطريقة التالية :

١ - حدد اليوم من السنة الحالية بالتقريب كما يلى :

```
n := trunc(30.42*(mm-1)) + dd
```

٢ - إذا كان mm = 2 ( شهر فبراير ) أضف 1 إلى قيمة n .

٣ - إذا كان mm > 2 ، وكانت mm < 8 ( شهر مارس أو أبريل أو مايو أو يونيو أو يوليو ) أنقص 1 من قيمة n

٤ - إذا كان mm > 2 , yy MOD 4 = 0 ( سنة كبيسة ) أضف 1 إلى قيمة n .

٥ - أضف 1461 إلى قيمة n لكل دورة من أربع سنوات بعد أول يناير 1960 .

٦ - أضف 365 إلى قيمة n لكل سنة إضافية بعد إتمام آخر دورة من أربع سنوات ، ثم أضف 1 ( لأحدث سنة كبيسة ) .

دعنا نستخدم دالة اسمها daysbeyond 1960 لتحديد قيمة مناسبة لـ n ، وذلك بمعرفة قيم mm dd yy . نفرض أن yy , dd, mm , n متغيرات مدى جزئى صحيحة . وفيما يلى دالة كاملة مبنية على الست خطوات سالفة الذكر .

```
FUNCTION daysbeyond1960(mm : month; dd : day; yy : year) : numberofdays;
(* This function converts a given date (month,day,year)
                    to the number of days beyond January 1, 1960. *)
VAR n : numberofdays;
```

```
BEGIN
   n := trunc(30.42*(mm-1)) + dd;
   IF mm=2 THEN n := n + 1;
   IF (mm > 2) AND (mm < 8) THEN n := n - 1;
   IF (yy MOD 4 = 0) AND (mm > 2) THEN n := n + 1;
   IF (yy-1960) DIV 4 > 0 THEN n := n + 1461*((yy-1960) DIV 4);
   IF (yy-1960) MOD 4 > 0 THEN n := n + 365*((yy-1960) MOD 4) + 1;
   daysbeyond1960 := n
END;
```

ومن السهل الآن إكمال بقية محتويات البرنامج . لعمل ذلك ... فإننا نعرف أنواع المدى الجزئى الصحيحة day , month , year , numberofdays ، ثم نوضح مجموعة من المتغيرات الشاملة من نوع المدى الجزئى ( days2 , days1 , yy , mm , dd ) . ويلى ذلك توضيح الدالة ، ثم يليه المجموعة الرئيسية التى تقرأ التاريخين وتتصل بالدالة لكل تاريخ منهما ، ثم تكتب النتيجة النهائية

وفيما يلى محتويات البرنامج .

```
PROGRAM dates(input,output);

(* THIS PROGRAM DETERMINES THE NUMBER OF DAYS BETWEEN TWO DATES *)

TYPE day = 1..31;
     month = 1..12;
     year = 1960..2100;
     numberofdays = 0..maxint;
VAR dd : day;
    mm : month;
    yy : year;
    days1,days2 : numberofdays;

FUNCTION daysbeyond1960(mm : month; dd : day; yy : year) : numberofdays;
(* This function converts a given date (month,day,year)
                 to the number of days beyond January 1, 1960. *)
VAR n : numberofdays;
BEGIN
   n := trunc(30.42*(mm-1)) + dd;
   IF mm=2 THEN n := n + 1;
   IF (mm > 2) AND (mm < 8) THEN n := n - 1;
   IF (yy MOD 4 = 0) AND (mm > 2) THEN n := n + 1;
   IF (yy-1960) DIV 4 > 0 THEN n := n + 1461*((yy-1960) DIV 4);
   IF (yy-1960) MOD 4 > 0 THEN n := n + 365*((yy-1960) MOD 4) + 1;
   daysbeyond1960 := n
END;

BEGIN   (* main action block *)
   write(' Enter the first date  (mm dd yyyy) :');
   readln(mm,dd,yy);
   days1 := daysbeyond1960(mm,dd,yy);
   write(' Enter the second date (mm dd yyyy) :');
   readln(mm,dd,yy);
   days2 := daysbeyond1960(mm,dd,yy);
   writeln;
   writeln(' There are ',days2-days1-1,' days between the two dates.')
END.
```

لاحظ أنه افترض أن التاريخ الثانى أكبر من التاريخ الأول . وينتج عن تنفيذ هذا البرنامج المخرجات التالية إذا ماكان تاريخا المدخلات هما 1963 29 12 و 1983 21 3 ( استجابة المستفيد موضوع تحته خط ) .

```
Enter the first date  (mm dd yyyy) :12 29 1963
Enter the second date (mm dd yyyy) : 3 21 1983

There are      7021 days between the two dates.
```

ويوضح المثال السابق استخدام بيانات المدى الجزئى فى مشكلة برمجة فعلية . أما استخدام البيانات من النوع المتعدد ، فموضح فى المثال التالى .

## مـثال (٨-١٠)

*برج هانوى* . اعتبر مرة أخرى المشكلة المذكورة في مثال (٢٦) فى الفصل السابع . دعنا نعدل الآن برنامج البسكال ، بحيث يستخدم بيانات من النوع المتعدد فى تمثيل الثلاثة أعمدة . وبصفة خاصة دعنا نقدم بيانات من النوع المتعدد poles ، والتى تتكون من ثلاثة عناصر ، هى : left , center , right . وفيما يلى البرنامج كاملا لهذه المشكلة :

```
PROGRAM towersofhanoi(input,output);

(* THIS PROGRAM SOLVES A WELL-KNOWN GAME USING
   RECURSIVE PROCEDURE CALLS AND USER-DEFINED DATA *)

TYPE poles = (left,center,right);
     disks = 0..maxint;
VAR n : disks;

   PROCEDURE transfer(n : disks; origin,destination,other : poles);
   (* Transfer n disks from the origin to the destination *)

      PROCEDURE diskmove(origin,destination : poles);
      (* Move a single disk from the origin to the destination *)
      BEGIN
         write(' Move ');
         CASE origin OF
            left   : IF destination = center
                        THEN writeln('left to center')
                        ELSE writeln('left to right');
            center : IF destination = left
                        THEN writeln('center to left')
                        ELSE writeln('center to right');
            right  : IF destination = center
                        THEN writeln('right to center')
                        ELSE writeln('right to left')
         END   (* case *)
      END;   (* diskmove *)

   BEGIN   (* transfer *)
      IF n>0 THEN BEGIN
                     transfer(n-1,origin,other,destination);
                     diskmove(origin,destination);
                     transfer(n-1,other,destination,origin)
                  END
   END;   (* transfer *)

BEGIN   (* main action block *)
   write(' Enter the number of disks -> ');
   readln(n);
   writeln;
   transfer(n,left,right,center)
END.
```

وهذه الصيغة للبرنامج ليست بنظافة الصيغة المقدمة فى مثال ٧ -٢٦ بسبب القيد الموضوع على البيانات من النوع المتعدد بأنها لايمكن أن تظهر فى عبارة write أو عبارة writeln . لاحظ بصفة خاصة الحاجة إلى مكون CASE المرهق بعض الشئ فى الإجراء diskmove ، وإلا فإن تعريف البرنامج على أية حال يكون مباشرا ، حيث إن الأعمدة الفردية يمكن أن يشار إليها بمصطلحات طبيعية أكثر .

ويعطى تنفيذ البرنامج فى حالة أن n = 3 المخرجات التالية . ( استجابات المستفيد موضوع تحتها خط ) .

```
Enter the number of disks -> 3

Move left to right
Move left to center
Move right to center
Move left to right
Move center to left
Move center to right
Move left to right
```

## أسئلة للمراجعة : Review Questions

١ - ماهما الفئتان العامتان للبيانات من النوع البسيط فى البسكال ؟ وكيف يمكن تقسيم كل منهما ؟

٢ - كيف تعرف بيانات النوع المتعدد ؟

٣ - فى أى مفهوم تحتوى بيانات النوع المتعدد على ترتيب متسلسل ؟

٤ - أى المؤثرات يمكن استخدامها مع بيانات النوع المتعدد ؟

٥ - أى الدوال القياسية يمكن استخدامها مع بيانات النوع المتعدد ؟

٦ - كيف تعرف بيانات المدى الجزئى ؟ قارنها مع بيانات النوع المتعدد .

٧ - إلى أى نوع من أنواع البيانات البسيطة يمكن تطبيق مفهوم المدى الجزئى ؟ مانوع البيانات البسيطة غير المتوافقة مع مفهوم المدى الجزئى ؟

٨ - أى المؤشرات يمكن استخدامها مع بيانات المدى الجزئى ؟

٩ - أى الدوال القياسية يمكن استخدامها مع بيانات المدى الجزئى ؟

١٠ - ماهى العناصر التى توجد فى جزء توضيحات برنامج البسكال ؟ وفى أى ترتيب يجب أن تظهر هذه العناصر ؟

١١ - تحت أى ظروف يكون من المفيد دمج تعريف نوع المدى الجزئى مع توضيح المتغير المناظر ؟ هل يمكن عمل ذلك مع بيانات النوع المتعدد أيضا ؟

١٢ - هل يمكن استخدام عنصر بيانات من النوع المتعدد كمتغير تحكم فى مكون FOR ؟ وهل يمكن استخدام عنصر بيانات من نوع المدى الجزئى لنفس الغرض ؟

١٣ – هل يمكن استخدام عنصر بيانات من النوع المتعدد كقائم بالاختيار فى مكون CASE ؟ وهل يمكن استخدام عنصر بيانات من نوع المدى الجزئى لنفس الغرض ؟

١٤ – ماهى القيود الموجودة على استخدام عناصر بيانات يعرفها المستفيد فى التعبير ؟

١٥ – هل يمكن لعنصر بيانات يعرفه المستفيد أن يظهر فى عبارة تحديد ؟ ماهى القيود التى تطبق فى هذه الحالة ؟

١٦ – ماهى القيود الموضوعة على استخدام عناصر بيانات يعرفها المستفيد فى عبارات مدخلات ومخرجات ؟

١٧ – ماهى مزايا استخدام بيانات النوع المتعدد فى برامج البسكال ؟ وماهى عيوب ذلك ؟ وضح إجابتك .

١٨ – ماهى مزايا استخدام بيانات المدى الجزئى فى برامج البسكال ؟ وماهى عيوب ذلك ؟ وضح إجابتك .

## مسائل محلولة : Solved Problems

١٩ – اعتبر تعريف النوع التالى :

فيما يلى عدة تعبيرات بوليان تستخدم عناصر البيانات هذه . حدد قيمة كل تعبير .

| التعبير | القيمة |
|---|---|
| re > fa | – خطأ |
| do <= ti | – صحيح |
| pred(mi) = re | – صحيح |
| mi = succ(fa) | – خطأ |
| ord(do) < 1 | – صحيح |
| 3 = ord(pred(fa)) + 1 | – صحيح |

٢٠ – يوضح التخطيط الهيكلى التالى تعريف واستخدام بيانات من النوع المتعدد ، أو من نوع المدى الجزئى .

(*a*)
```
PROGRAM sample1(input,output);
TYPE flavors = (vanilla,chocolate,strawberry,cherry,coconut);
     sizes = (small,medium,large);
VAR cone,dish,sundae : flavors;
    pint,quart : vanilla..strawberry;
    conesize : sizes;
BEGIN
   .
   .
   cone := cherry;
   .
   .
   IF quart = chocolate THEN . . .
                        ELSE . . .;
```

```
    CASE conesize OF
       small  : . . .;
       medium : . . .;
       large  : . . .
    END;
    .
    .
 END.
```

(*b*)
```
PROGRAM sample2(input,output);
TYPE primary = (yellow,cyan,magenta);
VAR color : primary;

   PROCEDURE proc1(VAR hue : primary);
   BEGIN
      .
      .
      IF hue = cyan THEN . . .
                    ELSE . . .;
      .
      .
   END;

BEGIN   (* main action statements *)
   .
   .
   FOR color = yellow TO magenta DO . . .
   .
   .
END.
```

(*c*)
```
PROGRAM sample3(input,output);
TYPE digits = '0'..'9';
VAR d1,d2 : digits;
BEGIN
   .
   .
   REPEAT
      .
      .
      .
   UNTIL d1 = d2;
   .
   .
END.
```

لاحظ أن المتغيرات d1 , d2 لاتمثل كميات عددية .

## مشاكل متكاملة : Supplementary Problems

٢١ – اعتبر تعريف النوع التالى :

```
TYPE cities = (Boston,Miami,Pittsburgh,Chicago, Denver,Phoenix,Seattle);
```

حدد قيمة كل تعبير من التعبيرات التاليه :

(*a*) pred(Pittsburgh)

(*b*) succ(Denver)

(*c*) ord(Seattle)

(*d*) ord(succ(Chicago))

(*e*) succ(ord(Chicago))

(*f*) Miami < Seattle

(*g*) Chicago < Pittsburgh

٢٢ – فيما يلى تخطيطات هيكلة لبرامج بسكال ، بعضها مكتوب بطريقة خاطئة . عرف الأخطاء .

(*a*)
```
PROGRAM sample(input,output);
TYPE days = (sun,mon,tues,wed,thurs,fri,sat);
VAR day1 : mon..fri;
    day2 : sun..sat;
BEGIN
   .
   .
   day1 := mon;
   WHILE day1 <> day2 DO
      BEGIN
         .
         .
         day1 := succ(day1)
      END;
   .
   .
END.
```

(*b*)
```
PROGRAM sample(input,output);
VAR highlight,foreground,background : colors
    main : red..blue;
CONST border = red;
TYPE colors = (red,green,blue,white,black);
BEGIN
   .
   .
   .
   foreground := 2;
   background := pred(foreground)
   .
   .
END.
```

(*c*)
```
PROGRAM sample(input,output);
TYPE suits = (clubs,diamonds,hearts,spades);
     values = 1..12;
VAR cardtype : suits;
    cardvalue : values;
    count,n : 1..maxint;
BEGIN
   .
   .
   write(' How many cards? ');
   readln(n);
```

```
   FOR count := 1 TO n DO
      BEGIN
         .
         .
         cardtype := . . .;
         cardvalue := . . .;
         writeln(count,cardtype,cardvalue)
      END
END.
```

(*d*)
```
PROGRAM sample(input,output);
TYPE suits = (clubs,diamonds,hearts,spades);
     values = 1..12;
VAR cardtype : suits;
    cardvalue : values;
    count,n : 1..maxint;
BEGIN
   .
   .
   write(' How many cards? ');
   readln(n);
   .
   .
   FOR count := 1 TO n DO
      IF cardvalue > 9 THEN writeln('Picture card');
   .
   .
END.
```

(*e*)
```
PROGRAM sample(input,output);
TYPE wines = (chablis,sauterne,rose,burgundy,chianti);
VAR dinner : wines;
    cost : real;

   FUNCTION special(cost : real) : wines;
   BEGIN
      .
      .
      .
      special := . . .
   END;

BEGIN   (* main action statements *)
   .
   .
   readln(cost);
   dinner := special(cost);
   .
   .
END.
```

## مشاكل مبرمجة : Programming Problems

٢٣ – عدل البرنامج الموضح فى مثال (٨ – ٩) بحيث إنه يقبل المعلومات التالية كمدخلات .

أ – تاريخ اليوم .

ب – اسم المستفيد ( افرض أن الاسم مكون من خمسة حروف ) .

ج – تاريخ ميلاد المستفيد .

ثم يحدد بعد ذلك عدد الأيام التى عاشها المستفيد .

( أدخل التواريخ بنفس الطريقة المذكورة فى مثال (٨ – ٩) )

٢٤ – عدل البرنامج الموضح فى مثال (٨ – ٩) ، بحيث إنه يقبل أى تاريخ بعد أول يناير 1960 كمدخلات ، ثم يطبع اليوم المناظر فى الأسبوع . ملاحظة : ١ يناير 1960 كان يوم جمعة Friday

٢٥ – عدل البرنامج الموحد فى مثال ( ٨ –١٠) بحيث إنه يمكن لمجموعة الأقراص أن تكون فى أى عمود ( ولا يلزم أن تكون فى العمود الموجود على أقصى اليسار ) ويمكن أن يكون العمود النهائى أى عمود من الأعمدة ( وليس بالضرورة أن يكون العمود الموجود على أقصى اليمين ) . حدد عمود الأصل وعمود المقصد كمؤشرات مدخلات . بعد ذلك اطبع الحركات المطلوبة لنقل الأعمدة من الأصل المجدد إلى الغاية المحددة .

٢٦ – عدل كل من البرامج الآتية ، مستخدما بيانات من النوع المتعدد ، أو من نوع المدى الجزئى ، مع استخدام إجراءات ودوال أيضا كلما كان ذلك ممكنا .

أ – إعادة حساب متوسط قائمة أعداد ( انظر مثال (٦ – ٩ ) .

ب – حساب قائمة مضروبات ( انظر مثال ( ٦ –١٠) .

ج – حساب الاستهلاك ( انظر مثال ( ٧ –١٢) .

د – محاكاة مباراة ( انظر مثال ( ٨ –١٨) .

٢٧ – حل كل مشكلة من مشاكل البرمجة التالية ، مستخدما بيانات من النوع المتعدد ، وبيانات من نوع المدى الجزئى كلما كان ذلك ممكنا ، تأكد أيضا من استخدام الأجزاء فى البرنامج ، وذلك باستخدام الإجراءات والدوال استخداما مناسبا .

أ – حساب أول n عدد من أعداد فيبوناكى كما سبق ذكر ذلك فى المشكله رقم ٥ – هـ فى الفصل ٦ . ثم تحديد أى هذه الأعداد عددا أوليا ( انظر المشكلة رقم ٥ – و الفصل ٦ ) .

ب – تحويل عدد صحيح موجب إلى عدد رومانى كما هو مذكور فى المشكلة رقم (٩) ٥٠ الفصل ٦ .

ج – برمجة مباراة متداخلة للعبة tic - tac - toe كما هو مذكور فى المشكلة رقم ( i ) ٥٠ الفصل ٧ .

# الفصل التاسع

# المنظومات

# Arrays

سبق أن أشرنا إلى أن البسكال يقدم العديد من فئات مختلفة لأنواع البيانات : بيانات بسيطة وبيانات مرتبة structured وبيانات مشيرات pointers . وحتى الآن اختصت مناقشتنا - على أية حال - بالبيانات بسيطة النوع . وقد رأينا أن البيانات بسيطة النوع تحتوى على أربعة أنواع : بيانات قياسية ، وهى البيانات الصحيحة integer ، والحقيقية real ، والحرفية char ، والبوليان boolean ( انظر الفصل الثالث ) كما أنها تحتوى على نوعين من البيانات التى يعرفها المستفيد أيضا ، وهى البيانات المتعددة enumerated ، وبيانات المدى الجزئى subrange ( انظر الفصل الثامن ) . وجميع البيانات البسيطة لها خاصية مشتركة ، وهى أن كل متغير يمثل عنصر بيانات واحد .

ونعيد الآن انتباهنا إلى البيانات مرتبة النوع . وفى واقع الأمر ... كل فصل من الفصول الأربعة القادمة يهتم بنوع مختلف من أنواع البيانات المرتبة . وتغطى هذه الفصول المنظومات arrays ، والسجلات records ، والملفات files والفئات sets على التوالى . وجميع هذه الأنواع من البيانات لها خاصية مشتركة ، وهى أنه يمكن لمعرف واحد أن يمثل عناصر بينات متعددة . كما يمكن أيضا الاتصال بعنصر البيانات الفردى بمفرده ( بالرغم من حدوث ذلك بطريقة مختلفة لكل نوع من أنواع البيانات المرتبة ) . وعلى هذا ... فعناصر البيانات المرتبة يمكن أن تعامل مع بعضها ، أو كل عنصر بمفرده ، وذلك طبقا لمتطلبات كل تطبيق خاص .

## ١ - منظومات ذات بعد واحد : 1. ONE - DIMNTIONAL ARRAYS

يمكن التفكير فى المنظومة ذات البعد الواحد على أنها قائمة list ( أو عمود ) بعناصر البيانات ، وكلها من نفس النوع . ويشار إليها جميعها باسم واحد . وكل عنصر فردى فى المنظومة ( أى كل عنصر من عناصر البيانات ) يمكن أن يشار إليه بتحديد اسم المنظومة ، يليه فهرس index (والذى يسمى دليل subscript ) محصورا بين قوسين مربعين . ويمكن لعناصر المنظومة أن تكون من أنواع البيانات ، طالما أنها جميعها من نفس النوع . وعلى هذا ... إذا كانت القائمة list منظومة ذات بعد واحد تحتوى على ٧١ عنصراً ، فإن العناصر الفردية للمنظومة تكون list [n] ... , list [2] , list [1] . وعلى هذا ... يمكن تمثيل محتويات المنظومة كمايلى :

$$
\text{list} = \begin{bmatrix} \text{list [1]} \\ \text{list [2]} \\ \cdot \\ \cdot \\ \cdot \\ \text{list [n]} \end{bmatrix}
$$

## مـثال (٩-١)

يحتوى برنامج بسكال على منظومة من النوع الحقيقى بها 100 عنصر . كل عنصر فردى له قيمة تساوى 0.01 مضروبة فى قيمة الفهرس الخاص به . وعلى هذا ... فإن :

```
list [1] = 0.01
list [2] = 0.02
list [3] = 0.03
        .
        .
        .
etc.
```

العنصر i فى list يمكن أن يشار اليه بأنه list [i] ، حيث i متغير صحيح . وعلى هذا ... تحدد قيمة 5 للمتغير i إذا ما أشار i الى العنصر الخامس فى المنظومة ، وهو list [5] . وقيمة هذا العنصر هى 0.05 .

وأسهل طريق لتعريف منظومة هو وضعها فى توضيح متغير كمايلى :

VAR *array name* : ARRAY [*index type*] OF *type*

ويمكن لنوع الفهرس أن يكون عددا ترتيبيا من النوع البسيط ( أى رقمًا صحيحًا أو حرفيًا أو بولـيان أو متعددًا ) أو من نوع المدى الجزئى . أما المنظومة نفسها ، فيمكن أن تكون من أى نوع ، بما فى ذلـك الأنـواع المرتـبة ( سوف يذكر المزيد بعد ذلك عن هذا ) . بالرغم من أن معظم التطبيقات تستخدم منظومات بسيطة النوع فقط . ( كنوع من الحاجة العملية . لاحظ أن المنظومة يمكنها أن تكون من النوع الحقيقى ، إلا أن فهرسها لايمكن أن يكون كذلك ) .

## مـثال (٩-٢)

افرض أن برنامج البسكال يحتوى على منظومة اسمها list مكونة من 100 عنصر . يمكن كتابة توضيح المنظومة كمايلى :

```
VAR list : ARRAY [1..100] OF real;
```

كما يمكن كتابة توضيحا بديلا كمايلى :

```
TYPE index = 1..100;
VAR list : ARRAY [index] OF real;
```

وعادة ماتكون الصيغة الأولى أفضل ، نظرا لبساطتها .

لاحظ أن عناصر المنظومة من النوع الحقيقى . أما الفهرس ، فهو من نوع المدى الجزئى الصحيح .

يمكن استخدام العناصر الفردية للمنظومة فى تعبيرات أو عبارات تحديد و عبارات write , read وخلافه ، حيث أنها متغيرات معتادة بسيطة النوع . ولعمل ذلك يجب أن يكتب عنصر المنظومة على أنه اسم المنظومة ، تتبعه قيمة مناسبة للفهرس ، محصورة بين قوسين مربعين . ويمكن التعبير عن قيمة الفهرس كثابت أو متغير أو تعبير . وعلى أية حال ... يجب أن يكون الفهرس من النوع الصحيح ( وليس الخاطئ ) كما يجب أن يقع داخل المدى الصحيح ( وليس الخاطـئ) للمنـظومة .

## مــثال (٩-٣)

دعنا نعتبر مرة أخرى منظومة list ذات البعد الواحد ، ومن النوع الحقيقى المذكورة فى المثال السابق . افرض أننا نريد الآن فحص قيمة كل عنصر من عناصر المنظومة ، وكتابة القيم التى تكون سالبة . يمكن تحقيق ذلك بمكون الدورة التالى :

```
FOR count := 1 TO 100 DO
    IF list [count] < 0
       THEN writeln(count:3, list [count]:4:1);
```

حظ أن قيمة الفهرس ( أى المتغير صحيح النوع count ) تكتب على نفس السطر مع عنصر المنظومة السـالب . وعلى هذا ... إذا كان ثالث عنصر من عناصر المنظومة له القيمة 5.0- ، فينتج عن ذلك سطر المخرجات التالى :

```
3 -5.0
```

وفى هذا المثال من الضرورى ألا تقع قيم count خارج المدى من 1 إلى 100 ، حيث إن فهرس المنظومة غير معرف خارج هذا المدى . ويمكننا – على أية حال – العمل مع مدى جزئى أصغر إذا ماأردنا ذلك . وعلى هذا ... كان يمكننا كتابة :

```
FOR count := 25 TO 50 DO
    IF list [count] < 0
       THEN writeln(count:3, list [count]:4:1);
```

ويجب أن يتبع استخدام عناصر المنظومة فى عبارات تحديد أو فى تعبيرات نفس القواعد السارية على المتغيرات بسيطة النوع . وعلى هذا ... إذا ماظهر عنصر منظومة فردى فى أى ناحية من عبارة تحديد ، فيجب أن يكون نوعا متوافقا مع المتغير الآخر أو التعبير الموجود فى هذه العبارة . وكذلك إذا مااستخدم فى تعبير ، فيجب أن يكون عنصر المنظومة من نوع متوافق مع بقية عناصر البيانات التى تظهر فى هذا التعبير .

## مــثال (٩-٤)

*الانحراف عن المتوسط* . افرض أننا نريد قراءة قائمة من n كمية حقيقية ، ونحسـب متوسـطها كـما فى مثال٦ – ٨ . بالإضافة إلى ذلك ... فإننا نريد حساب الانحراف أيضا لكل عدد من المتوسط باستخدام العلاقة :

```
deviation = x[i] - average
```

حيث x [i] يمثل كل عدد من الأعداد المعطاه ، ويمثل average المتوسط المحسوب .

لاحظ أننا نخزن كل عدد من الأعداد المعطاه فى منظومة حقيقية النوع وذات بعد واحد . وهذا جزء ضرورى من هذا البرنامج . والسبب فى ذلك – والذى يجب أن يكون مفهوما جيدا – هو مايلى :

فى كل الأمثلة السابقة التى حسبنا فيها المتوسط لقائمة أعداد ، كان كل عدد يحل محل العدد السابق له فى القائمة . وعلى هذا ... فقد كانت الأعداد الفردية غير متاحة لأى حسابات أخرى بعد الانتهاء من حساب المتوسط ، إلا أن هذه القيم الفردية يجب أن تحفظ الآن لحساب الانحراف المناظر ، وذلك بعد تحديد المتوسط . وعلى هذا ... فإنها تخزن فى منظومة x لها بعد واحد .

دعنا نقيد حجم x إلى 100 عنصر . نحتاج على أية حال إلى استخدام كل العناصر المائة . بدلا من ذلك سوف نحدد العدد الفعلى للعناصر ، وذلك بتحديد كمية صحيحة موجبة ( لاتزيد عن 100 ) للمتغير n .

وفيما يلى برنامجا كاملاً بلغة البسكال لأداء ذلك .

```
PROGRAM deviations(input,output);

(* THIS PROGRAM READS IN A LIST OF n NUMBERS,
   CALCULATES THEIR AVERAGE, AND THEN COMPUTES
   THE DEVIATION OF EACH NUMBER ABOUT THE AVERAGE *)

VAR n,count : integer;
    sum,average,deviation : real;
    x : ARRAY [1..100] OF real;

BEGIN
   BEGIN (* Read in the numbers and calculate the average *)
      write(' How many numbers will be averaged? ');
      readln(n);
      writeln;
      sum := 0;
      FOR count := 1 TO n DO
         BEGIN
            write(' i= ',count:3,'     x= ');
            readln(x [count]);
            sum := sum + x [count]
         END;
      average := sum/n;
      writeln(' The average is ',average);
      writeln;
   END;   (* calculate average *)

   BEGIN (* Calculate the deviations about the average *)
      FOR count := 1 TO n DO
         BEGIN
            deviation := x [count] - average;
            write(' i= ',count:3,'     x= ',x [count]);
            writeln('     d= ',deviation)
         END
   END   (* calculate deviations *)
END.
```

لاحظ طريقة عمل أجزاء هذا البرنامج ، حيث لاتوجد إجراءات أو دوال فى البرنامج . وبدلا من ذلك ... تحتوى المجموعة الرئيسية على عبارتين مركبتين ، كل منهما لها هدفها المميز . فكل عبارة مركبة معرفة مع تعليقاتها الخاصة بها . وهذه الطريقة مريحة بالنسبة للبرامج البسيطة القصيرة مثل هذا البرنامج .

افرض أن البرنامج نفذ الآن باستخدام القيم العددية الخمس التالية :

x[1] = 3.0, x[2] = -2.0, x[3] = 12.0, x[4] = 4.4, x[5] = 3.5

يظهر إدخال البيانات المتداخل كمايلى ( استجابات المستفيد موضوع تحتها خط ) .

```
How many numbers will be averaged? 5

i=  1     x= 3.0
i=  2     x= -2.0
i=  3     x= 12.0
i=  4     x= 4.4
i=  5     x= 3.5
```

وبمجرد إدخال جميع البيانات ، سوف تظهر النتائج التالية :

```
The average is  4.1800000E+00

i=  1    x=  3.0000000E+00    d= -1.1800000E+00
i=  2    x= -2.0000000E+00    d= -6.1800000E+00
i=  3    x=  1.2000000E+01    d=  7.8200000E+00
i=  4    x=  4.4000000E+00    d=  2.2000000E-01
i=  5    x=  3.5000000E+00    d= -6.8000000E-01
```

ويمكن التوسع فى بعض التطبيقات باستخدام فهرس من النوع المتعدد كما هو موضح فى المثال التالى .

## مثال (٩-٥)

فيما يلى جزءًا من برنامج بسكال يعرف منظومة لها بعد واحد ، ومن النوع الصحيح ، ولها فهرس من النوع المتعدد .

```
TYPE color = (red,white,blue,yellow,green);
VAR index  : color;
    values : ARRAY [color] OF integer;
```

لاحظ أن القيم معرفة بأنها منظومة لها خمسة عناصر ، وكل عنصر يمثل قيمة صحيحة .

إذا ماأردنا كتابة قيمة كل عنصر ، فيمكن أن ندخل المكون التالى فى مكان ما فى البرنامج .

```
FOR index := red TO green DO writeln(values[index]);
```

ويمكن لعناصر المنظومة أن تكون من النوع المتعدد أيضا كما هو موضح فى المثال التالى .

## مثال (٩-٦)

فيما يلى انحرافا عن الموقف المذكور فى مثال ٩ - ٥ . افرض أن values هى منظومة بها 100 عنصر من النوع color . وعلى هذا ... فيمكن أن يحتوى برنامج البسكال على نوع التعريف وتوضيح المتغير التاليين :

```
TYPE color = (red,white,blue,yellow,green);
VAR values : ARRAY [1..100] OF color;
```

وكل عنصر من عناصر المنظومة يأخذ الآن قيمة واحدة من القيم red , green , yellow , blue , white . وقيم الفهرس تكون صحيحة ، وتتراوح من 1 إلى 100 .

هناك نوع آخر من التطبيقات يظهر بصورة متكررة ، محتويا على استخدام السلاسل . فى مثل هذه المواقف عادة مانمثل السلسلة على أنها منظومة ذات بعد واحد من النوع الحرفى . وهذه الطريقة موضحة فى المثال التالى .

## مثال (٩-٧)

*الأسماء والعناوين* . فيما يلى برنامج بسكال بسيطًا يسمح بإدخال الاسم والعنوان ، وتخزينهما داخل الكمبيوتر ، ثم كتابتهما بعد ذلك . نفترض أن كل من الاسم واسم الشارع والمدنية المناظرة لاتتعدى 60 خانة . وعلى

هذا ... فإننا نقدم ثلاث منظومات ، كل عنصر من عناصرها مكون من 60 خانة من النوع الحرفى . وأسماؤها هى name , city , street على التوالى . ولاتحتاج أى سلسلة على أية حال لأن تحتوى على 60 خانة . بدلا من ذلك سوف ندخل رموزا متتالية داخل كل منظومة ، حتى نصل إلى نهاية السطر . وتستخدم دالة بوليان القياسية eoln فى هذا الغرض . وعلى هذا ... فإننا نستمر فى قراءة الرموز من كل سطر حتى تعود الدالة eoln بقيمة true . ( لاحظ أن eoln تصبح false عند بداية سطر جديد ) .

سوف نعد أيضا عدد الرموز فى كل منظومة ، بحيث إننا نعرف عدد الرموز ( عناصر المنظومة ) التى تكتب فيما بعد فى البرنامج . دعنا نشير إلى عدادات هذه الرموز بأنها streetcount , namecount , citycount على التوالى .

وفيما يلى محتويات برنامج البسكال . لاحظ أن البرنامج يحتوى بالضرورة على إجرائين : readin الذى يتسبب فى إدخال بيانات المدخلات داخل الكمبيوتر ، و writeout الذى يخرج المخرجات . لاحظ الملقنات للاسم والشارع والمدينة الموجودة داخل readin .

```
PROGRAM namesandaddresses(input,output);

(* THIS PROGRAM READS IN AND THEN WRITES OUT
      A NAME AND ADDRESS (STREET AND CITY) *)

VAR count,namecount,streetcount,citycount : 0..60;
    name,street,city : ARRAY [1..60] OF char;

PROCEDURE readin;
(* Read a name and address into the computer *)
BEGIN
   write('Name:   ');
   count := 0;
   REPEAT
         count := count + 1;
         read(name [count])
   UNTIL eoln;
   namecount := count;
   readln;

   write('Street: ');
   count := 0;
   REPEAT
         count := count + 1;
         read(street [count])
   UNTIL eoln;
   streetcount := count;
   readln;

   write('City:   ');
   count := 0;
   REPEAT
         count := count + 1;
         read(city [count])
   UNTIL eoln;
   citycount := count;
   readln
END;   (* readin *)
```

(تكملة البرنامج فى الصفحة التالية)

```
PROCEDURE writeout;
(* Write a name and address out of the computer *)
BEGIN
   FOR count := 1 TO namecount DO write(name [count]);
   writeln;
   FOR count := 1 TO streetcount DO write(street [count]);
   writeln;
   FOR count := 1 TO citycount DO write(city [count])
END;   (* writeout *)


BEGIN   (* main action block *)
   readin;
   writeln;
   writeout
END.
```

وينتج عن تنفيذ هذا البرنامج حوار المدخلات التالى ( استجابات المستفيد موجود تحتها خط ) .

```
Name:   Susan H. Gottfried
Street: 129 Old Suffolk Drive
City:   Monroeville, PA 15146
```

وبعد الانتهاء من إدخال البيانات ، فإن البرنامج يكتب الاسم والشارع والمدينة على النحو التالى :

```
Susan H. Gottfried
129 Old Suffolk Drive
Monroeville, PA 15146
```

وأخيرا يجب أن يميز القارئ أنه يمكن استخدام هذا البرنامج كنقطة بداية للعديد من البرامج الأخرى الأكثر تعقيدا . فمثلا يتطلب العديد من تطبيقات الأعمال ( مثل قوائم البريد والرواتب وحسابات المدينين ) أسماء وعناوين وبيانات مرتبطة بها لإدخالها الكمبيوتر وتعديلها بطريقة معينة ، ثم كتابتها بعد ذلك .

## مــثال (٩-٨)

*تسجيل قائمة بالأعداد* . اعتبر المشكلة المشهورة الخاصة بإعادة ترتيب قائمة بها n عدد من الأعداد فى ترتيب لقيمتها الجبرية المتزايدة . ويكتب البرنامج بطريقة لايستخدم فيها تخزين كبير .

وعلى هذا ... يحتوى البرنامج على منظومة واحدة فقط ذات بعد واحد من النوع الحقيقى ، اسمها x ، والتى تسجل عنصرا واحدا فى نفس الوقت .

وتبدأ العملية بفحص محتويات المنظومة لتحديد أقل عنصر ، ثم استبدال هذا العنصر بأول عدد من أعداد المنظومة . ( عند ذلك يصبح أصغر عنصر فى بداية القائمة ) وبعد ذلك يتم فحص بقية الأعداد ، وعددها n-1 لتحديد أصغرها ، والذى يستبدل مع العدد الثانى فى القائمة وهكذا ، حتى يعاد ترتيب محتويات المنظومة . ويتطلب ذلك عدد n-1 من المسارات خلال المنظومة ، بالرغم من أن طول كل عملية فحص يقل بصفة دائمة من مسار للمسار التالى له .

ولكى يحدد أقل عدد فى كل مسار ، فإننا نقارن كل عدد فى المنظومة x [i] على التوالى مـع عــدد البدايــة x [100] ، حيث 100 هو متغير صحيح يستخدم فى تسمية tag أحد عناصر المنظــومة . إذا كــان x [i] أصــغر مــن x [100] ، فإننا نستبدل العنصرين معا ، وإلا فإننا نترك العددين فى موضعهما الأصلى . وبعد تطبيق هذه العملية على محتويات المنظومة ، يصبح أول عدد هو أصغر محتويات المنظومة ، ثم تكرر بعد ذلك هذه العملية عدد n-2 من المرات ، ليصبح إجمالى عدد المسارات n-1 (loc = 2;1 , ... ,n-1) .

والسؤال المتبقى الوحيد هو كيفية تبديل عنصرين . نخزن أولا تخزينا وقتيا قيمة x[loc] فى دليل للمستقبل ، ثم نحدد القيمة الحالية للمتغير x[i] فى x[loc] . وأخيرا نحدد القيمة الأصلية original value للمتغير x[loc] التى سبق تخزينها فى x[i] . وعند ذلك يكون التبادل قد حدث .

ويمكن تنفيذ ذلك فى إجراء بسكال يسمى interchange كما هو موضح أدناه :

```
PROCEDURE interchange;
(* Interchange array elements from smallest to largest *)
BEGIN
   FOR loc := 1 TO n-1 DO
      FOR i := loc + 1 TO n DO
         IF x [i] < x [loc] THEN
            BEGIN
               temp := x [loc];
               x [loc] := x [i];
               x [i] := temp
            END
END;
```

نفترض فى هذا الإجراء أن loc , i متغيرات صحيحة تتراوح قيمتها من 1 إلى 100 . كما نفترض أيضا أن temp متغير حقيقى يستخدم فى التخزين المؤقت للمتغير x[loc] . لاحظ أن الإجراء يستخدم مكون FOR - TO متداخل .

دعنا نعتبر الآن عملية البرنامج الشامل كما هى موضحة فى التخطيط التالى :

(١) أقرأ حجم المنظومة x داخل الكمبيوتر .

(٢) اقرأ عناصر المنظومة داخل الكمبيوتر ، وذلك ردا على ملقنات التداخل .

(٣) أعد ترتيب الأعداد داخل المنظومة بالاتصال بإجراء interchange .

(٤) اكتب القيم التى أعيد ترتيبها .

وفيما يلى برنامج البسكال .

```
PROGRAM reorder(input,output);
(* THIS PROGRAM REORDERS A ONE-DIMENSIONAL ARRAY
        OF REAL NUMBERS FROM SMALLEST TO LARGEST *)
VAR n,i,loc : 1..100;
    x : ARRAY [1..100] OF real;
    temp : real;
PROCEDURE interchange;
(* Interchange array elements from smallest to largest *)
BEGIN
   FOR loc := 1 TO n-1 DO
      FOR i := loc + 1 TO n DO
         IF x [i] < x [loc] THEN
            BEGIN
               temp := x [loc];
               x [loc] := x [i];
               x [i] := temp
            END
```

(تكملة البرنامج فى الصفحة التالية)

```
END;   (* interchange *)

BEGIN   (* main action block *)
   write('How many numbers are there? ');
   readln(n);
   writeln;
   FOR i := 1 TO n DO
      BEGIN
         write('x [',i:3,']= ? ');
         readln(x [i])
      END;
   interchange;
   writeln;
   writeln('Rearranged Data:');
   writeln;
   FOR i := 1 TO n DO
       writeln('x [',i:3,']= ',x [i]:4:1);
END.
```

ويبدأ تنفيذ هذا البرنامج بإدخال متداخل للبيانات ، مثل الموجود أدناه ، مع وضع خط تحت استجابة المستفيد .

```
How many numbers are there? 5

x [   1] = ? 4.7
x [   2] = ? -2.3
x [   3] = ? 12.9
x [   4] = ? 8.8
x [   5] = ? 6.0
```

يعيد البرنامج ترتيب الكميات الخمس المدخلة ، وينتج المخرجات التالية :

```
x [   1] = -2.3
x [   2] =  4.7
x [   3] =  6.0
x [   4] =  8.8
x [   5] = 12.9
```

## ٢ – منظومات متعددة الأبعاد : 2. MULTIDIMENTICNAL ARRAYS

يجب الا يقتصر مفهوم المنظومة على منظومات أحادية البعد فى البسكال ، فيمكن تعريف منظومات متعددة الأبعاد أيضا . وفى واقع الأمر ... مثل هذه المنظومات مفيدة جدا فى أنواع عديدة ومختلفة من التطبيقات .

لقد رأينا بالفعل أنه يمكن التفكير فى منظومة ذات بعد واحد كقائمة ( أى عمود واحد ) من عناصر البيانات . وتقدم المنظومة متعددة الأبعاد التوسع الطبيعى لهذه الفكرة . وعلى هذا ... فيمكننا أن نفكر فى منظومة مزدوجة البعد على أنها جدول من عناصر البيانات يحتوى على صفوف وأعمدة . وأول بعد ( أو أول فهرس ) يشير إلى رقم السطر ، أما الثانى بعد ( أو ثانى فهرس ) فيشير إلى رقم العمود . وبالمثل يمكن التفكير فى المنظومة ذات ثلاثة أبعاد بأنها مجموعة من الجداول مثل صفحات الكتاب .

افرض على سبيل المثال أن table هو منظومة ذات بعدين تحتوى على m صف و n عمود . العناصر الفردية للمنظومة هى :

```
[1,1], table [1,2], . . . , table [1, n], table [2,1],
table [2,2], . . . , table [2, n], . . . , table [m, 1],
table [m, 2], . . . , table [m, n]
```

وعلى هذا ... فيمكن رؤية محتويات المنظومة بالطريقة التالية :

```
        ┌ table [1,1]   table [1,2]   . . .   table [1,n] ┐
        │ table [2,1]   table [2,2]   . . .   table [2,n] │
        │ table [3,1]   table [3,2]   . . .   table [3,n] │
table = │                    .                            │
        │                    .                            │
        │                    .                            │
        └ table [m,1]   table [m,2]   . . .   table [m,n] ┘
```

وبغض النظر عن أبعاد المنظومة ، فدائما ماتحتوى المنظومة على مجموعة من عناصر البيانات من نفس النوع . ويمكن الإشارة إلى كل عناصر البيانات بمعرف واحد ( أى باسم مشترك للمنظومة ) . ويمكن الإشارة إلى عنصر فردى فى المنظومات ( أى عنصر بيانات فردى ) بتحديد اسم المنظومة ، تليه القيم المناسبة للفهارس ( أو للأدلة ) محصورة بين أقواس مربعة ومفصولة عن بعضها بفواصل .

مـثال (٩ - ٩)

يحتوى أحد برامج البسكال على منظومة ذات بعدين من النوع الحقيقى اسمها table . تمثل هذه المنظومة جدولا من الأعداد يحتوى على 60 صفًا ، و 150 عمودًا كحد أقصى . وعلى هذا ... فإن أول فهرس يتراوح من 1 إلى 60 بينما يتراوح الفهرس الثانى من 1 إلى 150 .

افرض أننا نريد الاتصال بعنصر المنظومة الموجود فى الصف الثالث والعمود السابع . يمكن الإشارة إلى هذا العنصر ببساطة بكتابة table [3,7] . وبالمثل يمكننا الإشارة إلى عنصر المنظومة الموجود فى الصف i والعمود j بأنه table [i, j] ، حيث i , j متغيرات صحيحة النوع تستخدم كفهارس . وعلى هذا ... إذا ماحددت 12 كقيمة للمتغير i ، وحددت 5 كقيمة للمتغير j ، فإن table [i,j] تشير إلى عنصر المنظومة الموجود فى العمود الخامس والصف الثانى عشر .

ويمكن ظهور تعريف المنظومة متعددة الأبعاد مع توضيح المتغيرات بنفس طريقة المنظومة ذات البعد الواحد . وعلى أية حال ... يجب تحديد فهرس منفصل لكل بعد من أبعاد المنظومة . وعلى هذا ... يمكن كتابة تعريف المنظومة كمايلى :

```
VAR array name : ARRAY [index 1 type, index 2 type, . . . ,
                        index n type] OF type
```

يمكن أن تكون أنواع الفهارس الترتيبية بسيطة ( أى صحيحة أو حرفية و بوليان أو متعددة ) أو من نوع المدى الجزئى . وليس هناك حاجة لأن تكون جميعها من نفس النوع ، إلا أن عناصر البيانات نفسها فى المنظومة يجب أن تكون من نفس النوع ، بما فى ذلك الأنواع المرتبة .

## مثال (٩-١٠)

اعتبر مرة أخرى المنظومة الحقيقية ذات البعدين المسماه table المذكورة فى المثال السابق . يمكن تعريف هذه المنظومة بالطريقة التالية :

```
VAR table : ARRAY [1..60, 1..150] OF real;
```

وفيما يلى طريقة أخرى للتعبير عن نفس التعريف :

```
TYPE index1 = 1..60;
     index2 = 1..150;
VAR  table : ARRAY [index1,index2] OF real;
```

وفيما يلى طريقة ثالثة للتعبير عن نفس التعريف :

```
CONST limit1 = 60;
      limit2 = 150;
TYPE  index1 = 1..limit1;
      index2 = 1..limit2;
VAR   table : ARRAY [index1,index2] OF real;
```

ومن الواضح أن التعريف الأول هو أبسط هذه التعريفات .

لاحظ أن كل من الفهرسين من نوع المدى الجزئى الصحيح ، إلا أن عناصر المنظومة نفسها من النوع الحقيقى .

ويتطلب العديد من التطبيقات التى تحتوى على استخدام منظومات متعددة الأبعاد دورات متداخلة ، بمعدل دورة لكل بعد من أبعاد المنظومة . وعلى هذا ... فالتطبيق الموجة ناحية تشغيل العناصر فى جدول يمكن أن يستخدم دورتين ، واحدة داخل الأخرى . ويمكن استخدام الدورة الخارجية فى تشغيل كل صف من الصفوف الفردية ، والدورة الداخلية فى تشغيل الأعمدة داخل كل صف . وطريقة تحقيق ذلك موضحة فى المثال التالى .

## مثال (٩-١١)

معاملة الجدول . دعنا نكتب الآن برنامج بسكال متداخلاً ، يقرأ جدولاً من أعداد حقيقية داخل الكمبيوتر ، ويخزنها فى منظومة ذات بعدين ، ويحسب مجموعة الأعداد الموجودة فى كل صف وفى كل عمود ، ثم يكتب الجدول ومعه مجموع الصفوف ومجموع الأعمدة التى تم حسابها .

table = منظومة حقيقية ذات بعدين ، تحتوى على جدول معطى ، والمجموع الذى تم حسابه .

nrows = متغير صحيح يحدد عدد الصفوف الموجودة فى الجدول .

ncols = متغير صحيح يحدد عدد الأعمدة فى الجدول .

row = عداد صحيح يحدد رقم الصف .

col = عداد صحيح يحدد رقم العمود .

نفرض أن حجم الجدول لن يتعدى ١٠ صفوف و ١٠ أعمدة . ونضيف صفًا إضافيا وعمودا إضافيا داخل المنظومة على أية حال ، وذلك لتخزين مجموع كل صف ومجموع كل عمود فيهما . وعلى هذا ... يوضع مجموع كل صف فى العمود الموجود على أقصى اليمين ( أى العمود رقم ncols+1 ) ، ويوضع مجموع كل عمود فى الصف الموجود فى قاعدة الصفوف ( أى الصف رقم nrows+1 ) . وعلى هذا ... فإننا نعرف table بأنه منظومة حقيقية ذات بعدين ، لها حد أقصى من عدد الصفوف ١١ ، ومن عدد الأعمدة ١١ .

دعنا نستخدم تكوين الأجزاء فى إعداد البرنامج .دعنا بصفة خاصة نكتب إجراءات منفصلة لقراءة البيانات داخل الكمبيوتر ، وحساب مجموع كل صف ومجموع كل عمود وكتابة الجدول النهائى . وهذه الإجراءات تسمى readinput , rowsums , columnsums , writeoutput على التوالى .

والمنطق المطلوب لكل إجراء واضح ومباشر ، بالرغم من أنه يجب ملاحظة أنه مطلوب دورة مزدوجة داخل كل إجراء من هذه الإجراءات . فلقراءة الجدول الأصلى داخل الكمبيوتر على سبيل المثال يجب أن نقدم الدورة المزدوجة التالية .

```
FOR row := 1 TO nrows DO
   BEGIN
      FOR col := 1 TO ncols DO read(table [row,col]);
      writeln
   END
```

( العبارة المكتوبة مطلوبة لكى ينتقل إلى السطر التالى ) . وبالمثل الدورة التالية مطلوبة لحساب مجموع كل صف .

```
FOR row := 1 TO nrows DO
   BEGIN
      table [row,ncols+1] := 0;
      FOR col := 1 TO ncols DO
          table [row,ncols+1] := table [row,ncols+1] + table [row,col]
   END
```

ويمكن استخدام مكونات دورات مزدوجة أخرى لحساب مجموع كل عمود ، ولكتابة الجدول النهائى .

وفيما يلى برنامج بسكال كاملاً لأداء ذلك :

```
PROGRAM table1(input,output);

(* THIS PROGRAM READS IN A TABLE OF NUMBERS,
   SUMS THE COLUMNS WITHIN EACH ROW,
   AND THEN SUMS THE ROWS WITHIN EACH COLUMN *)

VAR row,col : 1..11;
    nrows,ncols : 1..10;
    table : ARRAY [1..11, 1..11] OF real;
```

(تكملة البرنامج فى الصفحة التالية)

```
PROCEDURE rowsums;
(* Add the columns within each row *)
BEGIN
   FOR row := 1 TO nrows DO
      BEGIN
         table [row,ncols+1] := 0;
         FOR col := 1 TO ncols DO
             table [row,ncols+1] := table [row,ncols+1] + table [row,col]
      END
END;   (* rowsums *)

PROCEDURE columnsums;
(* Add the rows within each column *)
BEGIN
   FOR col := 1 TO ncols DO
      BEGIN
         table [nrows+1,col] := 0;
         FOR row := 1 TO nrows DO
             table [nrows+1,col] := table [nrows+1,col] + table [row,col]
      END
END;   (* columnsums *)

PROCEDURE readinput;
(* Read the elements of the table *)
BEGIN
   write(' How many rows? (1..10) ');
   readln(nrows);
   writeln;
   write(' How many columns? (1..10) ');
   readln(ncols);
   writeln;
   FOR row := 1 TO nrows DO
      BEGIN
         writeln(' Enter data for row no. ',row:2);
         FOR col := 1 TO ncols DO read(table [row,col]);
         writeln
      END
END;   (* readinput *)

PROCEDURE writeoutput;
(* Write out the table and the corresponding sums *)
BEGIN
   writeln(' Original table, with row sums and column sums:');
   writeln;
   FOR row := 1 TO nrows + 1 DO
      BEGIN
         FOR col := 1 TO ncols + 1 DO write(table [row,col]:6:1);
         writeln
      END
END;   (* writeoutput *)

BEGIN   (* main action block *)
   readinput;
   rowsums;
   columnsums;
   writeoutput
END.
```

افرض أن البرنامج يستخدم أعداد الجدول التالية فى التشغيل .

| | | | |
|---|---|---|---|
| 2.5 | −6.3 | 14.7 | 4.0 |
| 10.8 | 12.4 | −8.2 | 5.5 |
| −7.2 | 3.1 | 17.7 | −9.1 |

عند تنفيذ البرنامج يحدث الحوار التالى ( استجابة المستفيد موضوع تحتها خط )

```
How many rows? (1..10) 3

How many columns? (1..10) 4

Enter data for row no.  1
2.5 -6.3 14.7 4.0

Enter data for row no.  2
10.8 12.4 -8.2 5.5

Enter data for row no.  3
-7.2 3.1 17.7 -9.1
```

عند ذلك يحسب البرنامج مجموع كل صف ، ثم ينتج المخرجات التالية :

```
  2.5  -6.3  14.7   4.0  14.9
 10.8  12.4  -8.2   5.5  20.5
 -7.2   3.1  17.7  -9.1   4.5
  6.1   9.2  24.2   0.4   0.0
```

ويمكننا أن نرى من هذه المخرجات أن مجموع عناصر الصف الأول هى 14.9 ( وهى 2.5-6.3+14.7+4 =14.9 ) . وبالمثل مجموع عناصر العمود الأول هى 6.1 ( وهى 2.5+10.8-7.2 = 6.1 ) وهكذا

ولاتحتاج فهارس المنظومة متعددة الأبعاد أن تكون كلها من نفس النوع . والقيد الوحيد هو أن كل فهرس يكون من النوع الترتيبى البسيط .

### مثال (٩-١٢)

فيما يلى بعض توضيحات إضافية لمنظومة متعددة الأبعاد :

```
TYPE color = (red,white,blue,yellow,green);
     index = 1..100;
VAR sample : ARRAY [color,index] OF boolean;
```

وعلى هذا ... فأحد عناصر sample يمكن أن يكون sample [white,25] ، وعنصر آخر يمكن أن يكون sample [green,66] . وحيث إن sample هى منظومة من نوع بوليان ، فإن القيم المحددة للعناصر الفردية تكون بوليان أيضا ، أى أن قيمتها يجب أن تكون صحيحة true أو خاطئة false فقط .

تذكر أن المنظومة يمكن أن يكون لها أى نوع ، بما فى ذلك النوع المرتب . وعلى هذا ... فمن الممكن تعريف منظومة ذات بعد واحد وعناصرها نفسها تكون عبارة عن منظومات ذات بعد واحد ( أى أن كل العناصر لها نفس البعد ) ومن نفس النوع ، مثل المنظومة الأم . ونتيجة تعريف مثل هذه المنظومة تكافئ منظومة ذات بعدين معتادة .

**مثـــال (٩-١٣)**

دعنا نعرف منظومة حرفية ذات بعد واحد تسمى codes ، عناصرها عبارة عن منظومات حرفية ذات بعد واحد أيضا . نفرض أن codes بها 25 عنصرا كحد أقصى ، وأن كل عنصر عبارة عن منظومة بها 50 عنصرا ، يمكن كتابة تعريف المنظومة كمايلى :

```
VAR codes : ARRAY [1..25] OF ARRAY [1..50] OF char;
```

وهذا يكافئ تعريف منظومة حرفية ذات بعدين معتادة ، وعلى هذا ... فيمكننا كتابة تعريفها بطريقة بديلة كمايلى :

```
VAR codes : ARRAY [1..25, 1..50] OF char;
```

كل توضيح من هذين التوضيحين يعرف جدولاً به 25 صفًا على الأكثر ، وفى كل صف مالايزيد عن 50 عنصراً من النوع الحرفى .

العناصر الفردية للمنظومة يمكن الاتصال بها بطريقتين مختلفتين أيضا . فمثلا يمكن الاتصال بالعنصر الموجود فى الصف i والعمود j كمايلى :

```
code [i][j]
```

أو كما يلى :

```
code [i,j]
```

وبالمثل يمكن الاتصال بالصف i كمايلى :

```
code [i]
```

وعلى هذا ... فالعنصر الموجود فى الصف 12 والعمود 20 يمكن الاتصال به بكتابة code[12][20] ، أو كتابة code[12,20] . كما أن كل عنصر من الصف 12 يمكن الاتصال بها بكتابة code[12] أيضا .

وفى معظم التطبيقات يكون من المريح استغلال عناصر منظومة متعددة الأبعاد باستخدام الطريقة الأصلية كما هى مذكورة فى مثال ٩ -١١ ، بدلا من الطريقة سالفة الذكر .

## ٣ - عمليات تجرى على محتويات المنظومات : 3. OPERATIONS WITH ENTIRE ARRAYS

هناك عمليات معينة يمكن إجراؤها على محتوى المنظومة ، وتؤثر - على ذلك - على كل عناصر المنظومة بنفس الطريقة . يمكننا بصفة خاصة تعريف أنواع المنظومات وتحديد منظومات لمنظومات أخرى ، وتمرير منظومات كمؤشرات إلى إجراءات أو إلى دوال . دعنا نعتبر كل من هذه المعالم بشئ من التفصيل .

تستخدم بعض البرامج العديد من المنظومات المختلفة من نفس النوع ، ولها نفس الأبعاد . وفي مثل هذه الحالات يكون من المريح تعريف نوع واحد من أنواع المنظومات ، ثم توضيح المنظومات الفردية كمتغيرات من هذا النوع .

وفي كلمات أخرى ... يمكن التعبير عن تعريف نوع المنظومة كمايلي :

```
TYPE name = ARRAY [index 1 type, index 2 type, . . . ,
                                  index n type] OF type
```

حيث إن أنواع الفهرس ونوع البيانات المنظومة لها نفس المعنى ، كما في توضيح المتغير بأنه من نوع المنظـومة ( أنظر القسم رقم ٢ ) .

### مـثال (٩-١٤)

افرض أن group 1 و group 2 هما منظومتان من النوع الصحيح ، لهما بعدان ، وكل منهما به 25 صفا كحد أقصى ، و 50 عمودا كحد أقصى . يمكن تعريف هاتين المنظومتين بالطريقة التالية :

```
TYPE table = ARRAY [1..25, 1..50] OF integer;
VAR group1,group2 : table;
```

وعلى هذا ... فإن table يعرفه المستفيد بأنه من النوع المرتب ( منظومة صحيحة النوع ، ذات بعدين ، لها 25 صفا كحد أقصى ، و 50 عمودا كحد أقصى ، وذلك بالتحديد ) . ويوضح المتغيران group 1 و group 2 بانهما من نفس نوع table . وعلى هذا ... فإن كل من group 1 و group 2 مصفوفة من النوع الصحيح ، ذات بعدين ، ولها 25 صفا كحد أقصى ، و 50 عمودا كحد أقصى .

ولاتحتاج توضيحات المتغيرات أن يكون لها نفس مدى تعريف النوع . فمثلا يمكن تعريف النوع داخل المجموعة الرئيسية ، ويمكن أن يظهر توضيحات لمتغير منظومة معين فورا بعد ذلك . وتوضيحات المتغير المتبقية يمكن عندئذ أن تظهر في إجراء أو في دالة .

وفي بعض الأحيان يكون من المرغوب فيه تحديد كل عناصر منظومة واحدة لتناظر عناصر منظومة أخرى ، حيث تكون كلا من المنظومتين من نفس النوع ، ولهما نفس الأبعاد . ويمكن تحقيق ذلك بعبارة تحديد واحدة ، مع تجنب الحاجة لتحديد كل عنصر على انفراد داخل أحد أنواع مكونات الدورات .

### مثـال (٩-١٥)

اعتبر المنظومتين الصحيحتين group 1 و group 2 مرة أخرى ، والمعرفتين في المثال السابق . افرض أن كل عناصر المنظومة group 1 تم قراءتها بالفعل داخل الكمبيوتر ، وأننا نرغب في تحديد هذه العناصر لتناظر عناصر المنظومة group 2 . يمكن تحقيق ذلك ببساطة بكتابة مايلي :

```
group2 := group1;
```

لاحظ أن عبارة التحديد الفردية هذه تكافئ الدورة المزدوجة المتداخلة التالية :

```
FOR row := 1 TO 25 DO
    FOR col := 1 TO 50 DO
        group2 [row,col] := group1 [row,col];
```

ومن الواضح أن عبارة التحديد البسيطة أبسط كثيرا .

وعلى أية حال ... ليس من الممكن أن تحتوى تعبيرات عددية أو بوليان على كل محتويات المنظومة . وعلـــى هـــذا ... فإذا كانت c,b,a متغيرات منظومة ، فإن مثل العبارة التالية غير مسموح به .

```
c := a + b;
```

وبالمثل لايمكن قراءة جميع محتويات المنظومة داخل الكمبيوتر بعبارة read أو readln واحدة . وعلى هذا ... يكون مطلوبا مكون دورة لقراءة العناصر المتعددة للمنظومة . وبصفة عامة .... فإن نفس القيد يقع على كتابة محتويات المنظومة كلها بعبارة write أو عبارة writeln واحدة . سوف نرى استثناءا هاما لهذا القيد الاخير فى قسم ٥ من الفصل التاسع .

ويمكن تمرير المحتويات الكلية لمنظومة إلى إجراء أو إلى دالة كمؤشر . لاحظ على أية حال أن المؤشر الرسمى المناظر ( داخل الإجراء أو الدالة ) يجب أن يصاحبه نوع بيانات مميز . وفى هذه الحالة يكون النوع منظومة . وعلى هذا يجب أن يعرف نوع المنظومة صراحة داخل مجموعة تحتوى على الإجراء أو على الدالة .

والطريقة المريحة لتحقيق هذا المتطلب هى تعريف نوع منظومة شامل global array type كما سبق ذكره فى بداية هذا القسم . ويمكن بعد ذلك توضيح مؤشرات نوع المنظومة الفعلية والمؤشرات الرسمية المناظرة كأعداد لهذه المنظومة كما هو مطلوب فى البرنامج . وهذه الطريقة موضحة فى المثال التالى .

## مـــثال (٩-١٦)

*جمع جدولين من الأعداد* . افرض أننا نريد قراءة جدولين صحيحين داخل الكمبيوتر ، ونحسب مجموع العناصر المناظرة ، أى :

```
c[i,j] = a[i,j] + b[i,j]
```

ثم نكتب بعد ذلك جدولا جديدا يحتوى على هذه المجموعات . نفرض أن كل الجداول تحتوى على نفس عدد الصفوف والأعمدة ، والتى لاتتعدى 20 صفا و 30 عمودا .

دعنا نستخدم تعريفات المتغيرات التالية :

table = منظومة ذات بعدين ، صحيحة النوع ، لها 20 صفا كحد أقصى ، و 30 عمودا كحد أقصى .

a , b , c = منظومات ، كل منها له بعدان من نفس نوع المنظومة table.

nrows = متغير صحيح يحدد العدد الفعلى للصفوف فى كل جدول .

ncols = متغير صحيح يحدد العدد الفعلى للأعمدة فى كل جدول .

row = عداد صحيح يحدد رقم الصف .

col = عداد صحيح يحدد رقم العمود .

سوف نستخدم تكوين أجزاء تشبه التكوين المستخدم فى مثال ١١ - ٩ ، وبصفة خاصة فإننا نعرف إجراء لقراءة البيانات داخل أى جدول ، وإجراء آخر لتجميع كل العناصر ، وإجراء ثالث لكتابة جدول جديد يحتوى على المجموع المحسوب . دعنا نسمى هذه الإجراءات writcoutput , computcsums, rcadinput على التوالى .

والبرنامج الفعلى يكون مباشرا كما هو مبين أدناه :

```
PROGRAM table2(input,output);

(* THIS PROGRAM READS IN TWO TABLES OF INTEGERS,
   CALCULATES THE SUMS OF THEIR RESPECTIVE ELEMENTS,
   AND WRITES OUT THE RESULTING TABLE CONTAINING THE SUMS

TYPE table = ARRAY [1..20,1..30] OF integer;
VAR  a,b,c : table;
     row,nrows : 1..20;
     col,ncols : 1..30;

PROCEDURE readinput(VAR t : table);
(* Read the elements of one table *)
BEGIN
   FOR row := 1 TO nrows DO
      BEGIN
         writeln(' Enter data for row no. ',row:2);
         FOR col := 1 TO ncols DO read(t[row,col]);
         writeln
      END
END   (* readinput *)

PROCEDURE computesums(t1,t2 : table; VAR t3 : table);
(* Sum the elements of two similar tables *)
BEGIN
   FOR row := 1 TO nrows DO
       FOR col := 1 TO ncols DO
           t3[row,col] := t1[row,col] + t2[row,col]
END;   (* computesums *)

PROCEDURE writeoutput(t : table);
(* Write out the elements of a table *)
BEGIN
   writeln(' Sums of the elements:');
   writeln;
   FOR row := 1 TO nrows DO
      BEGIN
         FOR col := 1 TO ncols DO write(t[row,col]:4);
         writeln
      END
END;   (* writeoutput *)

BEGIN   (* main action block *)
   write(' How many rows? (1..20) ');
   readln(nrows);
   writeln;
   write(' How many columns? (1..30) ');
   readln(ncols);
   writeln;
```

(تكملة البرنامج فى الصفحة التالية)

```
      writeln(' First table:');
      writeln;
      readinput(a);
      writeln(' Second table:');
      writeln;
      readinput(b);
      computesums(a,b,c);
      writeoutput(c)
   END.
```

يحتوى هذا البرنامج على خواص عديدة يجب ملاحظتها . لاحظ أولا أن البرنامج يحتوى على تعريف منظومة تسمى table . كما أن هناك توضيحا لثلاثة متغيرات a , b , c داخل المجموعة الرئيسية على أنها منظومات من نفس نوع table . بالإضافة إلى ذلك تحتوى الإجراءات على مؤشرات من نوع المنظومة ، موضحة أيضا بأنها من نفس نوع table . لاحظ أن مؤشر نوع المنظومة t ، والموجود فى الإجراء readinput ، هو مؤشر متغير varible . وهذا يسمح لعناصر t أن تعود من الاجراء إلى المجموعة الرئيسية . ومن ناحية أخرى ... فإن متغير نوع المنظومة t الموجود فى writeoutput هو مؤشر قيمة value ، حيث إن عناصر t المنقولة إلى هذا الإجراء ليست فى حاجة إلى العودة إلى المجموعة الرئيسية .

الإجراء المتبقى computersums يستغل كل من مؤشرات القيمة من نوع المنظومة ، ومؤشرات المتغير من نوع المنظومة . وبصفة خاصة فان أول مؤشرين ( t2 , t1 ) هما مؤشرا قيمة ، حيث إنهما يمثلان بيانات مدخلات للإجراء . أما المؤشر الثالث ( t3 ) ، فهو مؤشر متغير ، حيث إنه يمثل معلومات يجب عودتها إلى المجموعة الرئيسية .

افرض الآن أن البرنامج يستخدم لجمع جدولى الأعداد التاليين .

الجدول الأول

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |

الجدول الثانى

| | | | |
|---|---|---|---|
| 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 |

ينتج عن تنفيذ البرنامج الحوار التالى أولا ، وذلك لإدخال البيانات ( مرة أخرى استجابات المستفيد موضوع تحتها خط ) .

```
How many rows? (1..20) 3

How many columns? (1..30) 4

First table:

Enter data for row no.  1
1 2 3 4

Enter data for row no.  2
5 6 7 8

Enter data for row no.  3
9 10 11 12
```

(تكملة البرنامج فى الصفحة التالية)

```
Second table:

Enter data for row no.  1
10 11 12 13

Enter data for row no.  2
14 15 16 17

Enter data for row no.  3
18 19 20 21
```

عند ذلك يجمع البرنامج كل العناصر ، وينتج المخرجات التالية .

```
Sums of the elements:

11  13  15  17
19  21  23  25
27  29  31  33
```

ومن السهل التأكد من صحة هذه القيم فى الواقع .

## ٤ - منظومات مضغوطة : 4. PACKED ARRAYS

يمكن تعريف بعض المنظومات ، بحيث إنها تستغل ذاكرة الكمبيوتر بكفاءة ، وذلك بضغط عناصر البيانات لتكون قريبة من بعضها البعض . وتسمح هذه السمة بتخزين كمية معلومات كبيرة فى كمية معينة من الذاكرة .

ولكى يستغل ضغط المنظومة يجب استخدام الكلمات المحجوزة PACKED ARRAY فى مواصفات النوع ، أى :

```
VAR array name : PACKED ARRAY [index 1 type,
        index 2 type, . . . , index n type] OF type
```

وهذه السمة مرتفعة الكفاءة مع عناصر منظومة من النوع الحرفى ، ونوع بوليان ، والنوع المتعدد ، أو مع بيانات من نوع المدى الجزئى .

ومن ناحية أخرى ... فإن الوفر فى التخزين الذى يتحقق من الضغط يمكن أن يتلاشى بسبب فقدان سرعة الحسابات . وعلى هذا ... فأى برنامج يستغل منظومة مضغوطة يحتوى على موازنة بين هذين العنصرين . وليس من الواضح لبرنامج معين أى طريقة هى الأفضل . وهذا التمييز يتغير بصفة عامة من تطبيق لآخر .

وهناك ظروف معينة يكون فيها الضغط مفيدا ، وذلك من وجهة نظر اقتصاديات التخزين ، ومن وجهة نظر سرعة الحسابات أيضا . ويميل هذا لأن يكون صحيحا فى البرامج التى تحتوى على عدد كبير من تحديدات منظومات ( أى تحديد العناصر من منظومة لأخرى ) أو استدعاء إجراء أو دالة تمر فيه المنظومات كمؤشرات قيمة .

### مثال (٩-١٧)

اعتبر مرة أخرى المنظومة ذات البعدين ، ومن نوع بوليان المسماه sample ، والمعرفة فى مثال ٩ -١٢ . إذا

ماأوضحنا الآن sample بأنها منظومة مضغوطة ، فإن توضيح المنظومة يظهر كمايلى :

```
TYPE color = (red,white,blue,yellow,green);
     index = 1..100;
VAR sample : PACKED ARRAY [color,index] OF boolean;
```

سوف يتسبب هذا التوضيح فى ضغط عناصر المنظومة sample فى مساحة ذاكرة أقل عن التوضيح المناظر الموجود فى مثال ٩ - ١٢.

يحتوى البسكال على إجرائين قياسيين ، هما : pack , unpack ، يستخدمان فى تحويل المنظومات غير المضغوطة إلى منظومات مضغوطة ، والعكس صحيح . وبصفة خاصة ... فإن الإجراء القياسى pack ينقل بعض العناصر أو كلها ، الخاصة بمنظومة غير مضغوطة إلى منظومة مضغوطة . وهذا التكوين هو :

pack(*unpacked array name*, *index*, *packed array name*);

حيث يشير index إلى أول عنصر فى المنظومة غير المضغوطة التى ستنتقل

وعند الاتصال بهذا الإجراء تستمر عملية النقل حتى تملأ كل عناصر المنظومة المضغوطة . ويجب أن تحتوى المنظومة غير المضغوطة ( منظومة المصدر ) على عناصر كافية بعد عنصر index ، بحيث إن كل العناصر داخل المنظومة المضغوطة ( منظومة الهدف ) يمكن ملؤها .

## مثال (٩-١٨)

اعتبر المنظومتين التاليتين :

```
VAR unpacked : ARRAY [1..20] OF char;
    packed   : PACKED ARRAY [1..20] OF char;
```

العبارة التالية :

```
pack(unpacked,1,packed);
```

تتسبب فى أن كل عنصر فى unpack ( بدءا بالموقع 1 ) ينقل إلى الموقع المناظر داخل pack . أى أن :

```
packed [1] := unpacked [1];

packed [2] := unpacked [2];



packed [20] := unpacked [20];
```

وينقل الإجراء القياسى unpack عناصر المنظومة المضغوطة إلى المنظومة غير المضغوطة ، بدءا بالعنصر index من المنظومة غير المضغوطة . والتكوين هو :

unpack(*packed array name*, *unpacked array name*, *index*);

وتستمر عملية النقل حتى تنقل كل عناصر المنظومة المضغوطة . ويجب أن يكون فى المنظـومة غير المضـغوطة ( أى منظومة الهدف ) عناصر كافية بعد العنصر index ، بحيث يمكن نقل كل العناصر داخـل المنظـومة المضـغوطة ( منظومة المصدر ) .

**مـثال (٩-١٩)**

اعتبر المنظومتين التاليتين :

```
VAR packed   : PACKED ARRAY [1..13] OF char;
    unpacked : ARRAY [1..20] OF char;
```

افرض أن packed تحتوى على الرموز الثلاثة عشر التالية :

```
North America
```

وعلى هذا ... ينتج عن العبارة التالية :

```
unpack(packed,unpacked,5);
```

التحديدات التالية :

```
unpacked [5] := packed [1];     (hence unpacked [5] = 'N')

unpacked [6] := packed [2];     (hence unpacked [6] = 'o')



unpacked [17] := packed [13];   (hence unpacked [17] = 'a')
```

أما بقية عناصر unpacked الأخرى ، فتبقى بدون تعريف .

وهناك قيد واحد يجب أخذه فى الاعتبار عند استخدام منظومات مضغوطة ، وهو أنه لايمكن أن تمرر العناصر الفردية لمنظومة مضغوطة كمؤشرات فعلية إلى إجراءات أو دوال ، إلا أن المحتوى الكلى يمكن - على أية حال - تمريره بالطريقة المعتادة .

## ٥ - السلاسل ومتغيرات السلاسل : 9. STRINGS AND STRING VARIABLES

مفهوم المنظومة المضغوطة مهم بصفة خاصة بالاتصال مع السلاسل ، حيث إن السلسلة التى بها n رمزا عادة ماتعتبر بأنها منظومة ذات بعد واحد مضغوطة ، وبها n عنصر ، ولها النوع الحرفى .

**مـثال (٩-٢٠)**

افرض أن أحد برامج البسكال يحتوى على تعريف الثابت التالى :

```
CONST title = 'The Super-Duper Computer Company';
```

يمثل المعرف title منظومة مضغوطة مكونة من 32 عنصرا من النوع الحرفي .

ويمتد مفهوم السلسلة ليشمل متغيرات مثل الثوابت . وعلى هذا ... فإن متغير من النوع :

```
PACKED ARRAY [1..n] OF char
```

يشار إليه بانه متغير سلسلة string variable . مثل هذه المتغيرات يمكن استخدامها لتمثيل سلاسل لها طول n .

## مثال (٩-٢١)

افرض أن برنامج بسكال يحتوى على توضيح المتغير التالي :

```
VAR heading : PACKED ARRAY [1..14] OF char;
```

المعرف heading هو متغير سلسلة ، حيث إنه يمثل منظومة حرفية مضغوطة بها 14 عنصرا .

ويشتمل البسكال القياسي على بعض اختيارات خاصة للمنظومات المضغوطة ، والتي يمكن استخدامها مع السلاسل أو متغيرات السلاسل فقط . فمثلا يمكن تحديد سلسلة بها n عنصرا لمتغير سلسلة ذى n عنصر ، وذلك باستخدام عبارة تحديد عادية . ( لاحظ أن هذه العملية الفردية تشمل تحديد لتسلسل عناصر السلسة ) .

## مثال (٩-٢٢)

اعتبر مرة أخرى المنظومة المضغوطة heading المعروفة فى المثال السابق ، حيث إن heading هو متغير سلسلة له 14 عنصرا ، فيمكننا أن نحدد أى سلسلة لها 14 عنصرا للمنظومة heading ، أى :

```
heading := 'Dallas Cowboys';
```

ويمكننا أن نحدد بالطبع متغير سلسلة واحد لمتغير آخر ، على أن يشتمل كلا من المتغيرين على نفس عدد العناصر . ( تذكر أن هذه السمة تسرى على كل المنظومات كما سبق ذكره فى القسم ٣ من هذا الفصل )

## مثال (٩-٢٣)

افرض الآن أن برنامج البسكال يحتوى على توضيح المتغير التالي :

```
VAR factory,warehouse : PACKED ARRAY [1..11] OF char;
```

يستخدم المعرفان warehouse , factory متغيرات سلسلة ، حيث إن كلا منهما يمثل منظومة مضغوطة حرفية . لاحظ أيضا أن كلا من المتغيرين معرف بأنه منظومة لها 11 عنصرا . وعلى هذا ... يمكن تحديد إحدى المنظومات للمنظومة الأخرى .

والآن افرض أن البرنامج يحتوى على تحديد السلسلة التالى :

```
factory := 'Seattle, WA';
```

وعلى ذلك ... ففى نقطة لاحقة فى البرنامج يمكننا أن نحدد نفس السلسلة لمتغير سلسلة آخر له ١١ عنصرا ، مـثل :

```
warehouse := factory;
```

وعلى هذا ... فإن warehouse يمثل أيضا السلسلة "Seattle,WA." .

ويمكن مقارنة السلاسل ذات الطول المتساوى مع بعضها بواسطة المؤشرات العلاقية . ويمكن اجراء مثل هذه المقارنات بين ثابتى سلسلة أو متغيرى سلسلة ، أو ثابت سلسلة ومتغير سلسلة . وفى كل حالة من هذا الحالات فإننا ننتج تعبير بوليان يكون صحيحا true أو خاطئا false .

وعند استخدام مؤثر علاقى لتوصيل سلاسل أو متغيرات سلاسل ، فإن المؤثرين = و < > يحددان التكافؤ وعدم التكافؤ على التوالى . وتشير المؤشرات الأربعة المتبقية > و = > و < و = < إلى ترتيب الرموز داخل كل سلسلة ، كما هو معرف بشفرة الرموز الخاصة المستخدمة .

## مـثال (٩-٢٤)

أعتبر توضيح متغير السلسلة التالى :

```
TYPE colors = PACKED ARRAY [1..5] OF char;
VAR  c1,c2,c3 : colors;
```

افرض الآن أن متغيرات السلسلة محدد لها قيم السلسلة التالية .

```
c1 := 'black';
c2 := 'white';
c3 := 'green';
```

فيما يلى عدة تعبيرات بوليان تشمل هذه المتغيرات ، ومعها القيم المناظرة لكل تعبير .

| القيمة | | تعبير بوليان |
|---|---|---|
| صحيح | ( لأن 'b' < 'w' ) | c1 < c2 |
| صحيح | ( لأن 'g' < 'w' ) | c3 < c2 |
| خطأ | ( لأن 'l' < 'r' ) | c1 > 'brown' |
| صحيح | ( لأن 'o' > 'e' ) | 'gross' > c3 |

ويجب أن يكون مفهوما أن استخدام كل محتوى المنظومة كعنصر يمكن التأثير عليه فى تعبير بوليان يكون صحيحا إذا ماكانت المنظومة تمثل متغير سلسلة فقط . مثل هذه المقارنات تسمح بأنواع أخرى من المنظومات .

وهناك سمة أخرى تصاحب متغيرات السلسلة ، وتشمل استخدام عبارات writeln , write . يمكن أن توجد متغيرات السلسلة داخل هذه العبارات ، متسببة على ذلك فى كتابة كل محتوى السلسلة ( أى كل عناصر السلسلة المضغوطة ) ، دون الحاجة إلى مكونات الدورة .

### مـثال (٩-٢٥)

فيما يلى جزءا من برنامج بسكال :

```
    .
    .
VAR factory : PACKED ARRAY [1..11] OF char;
BEGIN
    .
    .
    factory := 'Seattle, WA';
    .
    .
    writeln(' Factory Location: ',factory);
    .
    .
END.
```

لاحظ أن عبارة writeln تحتوى على ثابت سلسلة ومتغير سلسلة .

عند تنفيذ هذا البرنامج نحصل على المخرجات التالية .

```
Factory Location: Seattle, WA
```

ويجب أن يكون مفهوما أن متغير السلسلة هو نوع المنظومة الوحيد الذى يمكن أن يوجد فى عبارة writeln , write بهذه الطـريقة . والأكثر من هذا فالإمكانية الشبيهة لذلك غير ممكنة عند استخدام عبارة readln , read ، وذلك حتى مع متغيرات السلسلة . وعلى هذا ... فيجب إدخال السلاسل داخل الكمبيوتر رمزا رمزا باستخدام أحد أنواع مكونات الدورة .

### مـثال (٩-٢٦)

فيما يلى جزءا من برنامج بسكال .

```
    .
    .
VAR factory : PACKED ARRAY [1..11] OF char;
    i : 1..11;
BEGIN
    .
    .
    FOR i := 1 TO 11 DO read(factory[i]);
    writeln;
    .
    .
    writeln(' Factory Location: ',factory);
    .
    .
END.
```

لاحظ الفرق بين عبارات المدخلات والمخرجات فى هذا المثال . لاحظ أيضا الاختلافات بين هذا المثال والمثال السابق .

المقدرة على تعريف ومعاملة متغيرات السلسلة تفتح الباب لفئة واسعة من تطبيقات الكمبيوتر . والمثال التالى يعطى بعد التوضيح لذلك .

مثال (٩-٢٧)

منتج حروف لعبة *Pig Latin* : هى صيغة شفرة من الإنجليزية ، وعادة مايستخدمها الأطفال كإحدى ألعابهم . وتتكون كلمة latin من كلمة إنجليزية بنقل أول صوت ( عادة مايكون أول حرف ) إلى نهاية الكلمة ، ثم يضاف الحرف "a" وعلى هذا ... فكلمة cat تصبح "atca" ، وكلمة pascal تصبح "ascalPa" ، وكلمة pig latin تصبح "igpa atinla" وهكذا .

دعنا نكتب برنامج بسكال يقبل سطرا من نص مكتوب باللغة الإنجليزية ، ثم يطبع بعد ذلك السطر المناظر فى pig latin . نفترض أن كل رسالة نصية يمكن كتابتها على سطر به 80 خانة ، مع وجود فراغ واحد بين الكلمات المتتالية على السطر . ( نحتاج فى واقع الأمر أن لاتزيد رسالة pig latin عن 80 خانة . وعلى هذا ... فيجب أن تكون الرسالة الأصلية أقل من 80 خانة بعض الشئ ، حيث إن رسالة pig latin المناظرة تكون أطول بإضافة حرف a لكل كلمة من كلمات السطر ) . وللتبسيط ننقل أول حرف فقط ( وليس أول صوت ) من كل كلمة . كما أننا نهمل أيضا أى اعتبارات خاصة يمكن أن تعطى للحروف الكبيرة ولعلامات التنقيط .

وتحتوى عملية الحسابات الشاملة الخطوات الرئيسية التالية :

(١) وضع قيم ابتدائية لكل المنظومات ( متغيرات السلسلة ) ، وذلك بتحديد فراغات لكل العناصر .

(٢) قراءة المحتوى الكلى للسطر داخل الكمبيوتر ( عدة كلمات ) .

(٣) تحديد عدد الكلمات الموجودة فى السطر ( وذلك بعد عدد الفراغات الفردية التى يتبعها حرف ) .

(٤) اعادة ترتيب الكلمات فى صورة pig latin ، وذلك كلمة كلمة كما يلى :

(أ) تحديد نهاية كل كلمة .

(ب) نقل أول حرف إلى نهاية الكلمة .

(ج) إضافة حرف "a" إلى نهاية كل كلمة معدلة .

(٥) كتابة كل محتويات السطر الجديد .

ونستمر فى إعادة هذا الإجراء حتى يقرأ الكمبيوتر سطرا من نص ، تكون أول كلمة ( أو أول ثلاثة أحرف الأولى فيه ) هــى end .

ولتنفيذ هذا العمل ، فإننا نستخدم مشيرين "pointers" اسمهما p1 , p2 على التوالى . يحدد أول مشير p1 موقع بداية كلمة معينة داخل السطر الأصلى من النص . ويحدد المشير الثانى p2 نهاية الكلمة . لاحظ أن الرمز الموجود فى العمود الذى يسبق رقم العمود p1 يكون فراغا ( فيما عدا أول كلمة ) . كما أن الرمز الموجود فى العمود الذى يلى العمود رقم p2 يكون فراغا أيضا .

دعنا نستخدم طريقة الأجزاء مرة أخرى فى إعداد هذا البرنامج . قبل مناقشة كل جزء من الأجزاء ، فإننا نعرف متغيرات البرنامج التالية :

english = متغير سلسلة ( أى منظومة مضغوطة حرفية ) يمثل السطر الأصلى من النص .

latin = متغير سلسلة يمثل السطر الجديد من النص ( أى سطر pig latin ) .

flag = متغير بوليان يظل صحيحا true حتى يتم إدخال كلمة "end" ، وذلك ليسمح بتكرار تنفيذ البرنامج .

words = متغير من النوع الصحيح يحدد عدد الكلمات الموجودة فى السطر المعطى من النص .

n = متغير من النوع الصحيح يستخدم كعداد للكلمات ( n = 1 , 2 , ... ,words ) .

count = متغير من النوع الصحيح يستخدم كعداد للرموز داخل كل سطر من الأسطر ( count = 2,1 , ... , 80 ) .

وهذه المتغيرات بالإضافة بالطبع إلى المتغيرين الصحيحين p1 , p2 الذين سبق ذكرهما .

دعنا نعود الآن إلى التخطيط الشامل للبرنامج الذى سبق ذكره . يمكن أداء أول خطوة ( وهى وضع قيم ابتدائية للمنظومات ) بطريقة مباشرة طبقا للإجراء التالى .

```
PROCEDURE initialize;
(* initialize the arrays with blank spaces *)
BEGIN
   FOR count := 1 TO 80 DO
      english [count] := ' ';
   latin := english
END;
```

ويمكن أداء الخطوة الثانية بإجراء بسيط أيضا . ويحتوى هذا الاجراء على دورة شرطية ( مكون WHILE-DO) يستمر فى قراءة الرموز من النهاية الطرفية حتى تكتشف نهاية السطر ( أى حتى تصبح الدالة eoln صحيحة true ) . هذا التسلسل للرموز يصبح عناصرا لمتغير السلسلة english . وفيما يلى المحتويات الكلية للإجراء .

```
PROCEDURE readinput;
(* read one line of English *)
BEGIN
   count := 0;
   WHILE NOT eoln DO
      BEGIN
         count := count +1;
         read(english [count])
      END;
   readln
END;
```

الخطوة 3 من التخطيط الشامل للبرنامج مباشرة أيضا . ويمكننا أن نفحص ببساطة السطر لمعرفة ما اذا كان هناك خانة فارغة يتبعها حرف . عند ذلك تزداد الكلمة counter ( عداد words ) فى كل مرة يظهر فيها خانة فارغة . وفيما يلى جزء عد الكلمات :

```
PROCEDURE countwords;
(* scan the line and count the number of words *)
BEGIN
   words := 1;
   FOR count := 1 TO 79 DO
      IF (english [count] = ' ') AND (english [count + 1] <> ' ')
         THEN words := words + 1
END;
```

دعنا نعتبر الان الخطوتين رقم 4 ورقم 5 من التخطيط الشامل للبرنامج . وهذه الخطوات هى فى واقع الأمر قلب البرنامج . ويمكن دمجهما فى واقع الأمر فى إجراء واحد ، حيث إن الخطوة رقم 5 لاتتطلب إلا عبارة فردية واحدة فقط . ومن ناحية أخرى ... فإن الخطوة رقم 4 تتطلب ثلاث عمليات منفصلة ، بالرغم من ارتباطها مع بعضها .

ويجب أن نعرف أولا نهاية كل كلمة ، وذلك بإيجاد أول خانة فارغة بعد p1 . عند ذلك نحدد الرموز التى تتكون منها الكلمة فى pig latin ، وذلك بأخر حرف فى نهاية الكلمة . ويجب أن يؤدى ذلك بحرص ، حيث إن السطر الجديد من النص يكون أطول من السطر الأصلى ( وذلك بسبب حروف "a" الإضافية ) . وعلى هذا ... فإن الحروف فى أول كلمة pig latin تمثل المواقع من p1 وحتى p2+1 . والحروف فى ثانى كلمة تمثل المواقع من p1+1 إلى p2+1 ( لاحظ أن هذه هى قيم جديدة لكل من p2 , p1 ) وهكذا . ويمكن تعميم هذه القواعد كما يلى :

**أولا** ، تنقل كل حروف الكلمة رقم n ، فيما عدا الحرف الأول من السطر الأصلى إلى السطر الجديد . ويمكن تحقيق ذلك بكتابة .

```
FOR count := p1 TO p2 - 1 DO
    latin [count + (n - 1)] := english [count + 1];
```

اعتبر العبارة التالية :

```
latin [count + (n - 1)] := english [count + 1];
```

داخل أى كلمة تتسبب هذه العبارة فى أن ثانى حرف إنجليزى يصبح أول حرف pig latin ، وثالث حرف إنجليزى يصبح ثانى حرف pig latin وهكذا . وتستخدم الكمية ( n-1 ) فى تضبيط موقع كل كلمة pig latin بسبب الحروف الزائدة التى يتم إضافتها ( أى حروف a فى نهاية كل كلمة ) وعلى هذا ... فإن الكلمة الثانية ( n=1 ) تضبط ( أى ترحل إلى اليمين ) بإزاحتها خانة واحدة ، وذلك بسبب وجود a فى نهاية أول كلمة والكلمة الثالثة ( n=3 ) تضبط بإزاحتها خانتين فقط .

وبعد تنفيذ الدورة المذكورة أعلاه لكل الحروف الموجودة داخل الكلمة باستثناء الحرف الأول ، فإننا نضيف أول حرف يتبعه الحرف "a". ويتم تحقيق ذلك بالطريقة التالية :

```
latin [p2 + (n-1)] := english [p1];
latin [p2 + n] := 'a';
```

ثم نعيد وضع قيمة p1 إلى :

```
p1 := p2 + 2
```

وذلك للإعداد للكلمة التالية .

وتعاد هذه المجموعة من العمليات لكل كلمة من كلمات السطر الأصلى . وبعد ذلك يكتب السطر الجديد ، والذى يحتوى على المكافئ من pig latin للسطر الأصلى .

وفيما يلى إجراء تحقيق ذلك .

```
PROCEDURE rearrangewords;
(* rearrange each word into piglatin, then write out the entire line *)
BEGIN
   pl := 1;
   FOR n := 1 TO words DO
      BEGIN
         (* locate end of word *)
         count := pl;
         WHILE english [count]  <> ' ' DO count := count + 1;
         p2 := count - 1;

         (* transpose first letter and add 'a' *)
         FOR count := pl TO p2 - 1 DO
             latin [count + (n - 1)] := english [count + 1];
         latin [p2 + (n - 1)] := english [pl];
         latin [p2 + n] := 'a';
         pl := p2 + 2
      END;
   writeln(latin)
END;
```

دعنا نعتبر الآن الجزء الرئيسى . هذا الجزء من البرنامج لايزيد عن أن يكون عبارة عن رسالة ابتدائية يتبعها مكون REPEAT-UNTIL يسمح بتكرار تنفيذ البرنامج ( حتى تظهر كلمة end كأول ثلاثة أحرف فى النص الإنجليزى ) . ونبدأ بصفة خاصة بتحديد قيمة true لـ flag ، ثم نغير هذه القيمة إلى false عند ظهور كلمة "end" داخل الدورة . ويمكن أن يوضع مكون IF-THEN داخل الدورة لاختبار هذا الشرط . وعند ذلك يفصل إجراء الدورة بمجرد أن تصبح قيمة flag خاطئة flase .

وفيما يلى محتويات البرنامج . لاحظ إضافة العديد من التعليقات إلى الإجراءات التى سبق ذكرها .

```
PROGRAM piglatin(input,output);
(* THIS PROGRAM ACCEPTS A LINE OF ENGLISH TEXT AND CONVERTS IT TO PIGLATIN. *)
(* Word pointers: pl -> beginning of word
                  p2 -> end of word        *)
TYPE line = PACKED ARRAY [1..80] OF char;
VAR english,latin : line;
    pl,p2,count,n,words : integer;
    flag : boolean;
PROCEDURE initialize;
(* initialize the arrays with blank spaces *)
BEGIN
   FOR count := 1 TO 80 DO
      english [count] := ' ';
   latin := english
END;   (* initialize *)
PROCEDURE readinput;
(* read one line of English *)
BEGIN
   count := 0;
   WHILE NOT eoln DO
      BEGIN
         count := count + 1;
         read(english [count])
      END;
```

(تكملة البرنامج فى الصفحة التالية)

```
   readln
 END;   (* readinput *)

PROCEDURE countwords;
(* scan the line and count the number of words *)
BEGIN
   words := 1;
   FOR count := 1 TO 79 DO
      IF (english [count] = ' ') AND (english [count + 1] <> ' ')
         THEN words := words + 1
END;   (* countwords *)

PROCEDURE rearrangewords;
(* rearrange each word into piglatin, then write out the entire line *)
BEGIN
   p1 := 1;
   FOR n := 1 TO words DO
      BEGIN

         (* locate end of word *)
         count := p1;
         WHILE english [count]  <> ' ' DO count := count + 1;
         p2 := count - 1;

         (* transpose first letter and add 'a' *)
         FOR count := p1 TO p2 - 1 DO
             latin [count + (n - 1)] := english [count + 1];
         latin [p2 + (n - 1)] := english [p1];
         latin [p2 + n] := 'a';
         p1 := p2 + 2
      END;
   writeln(latin)
END;   (* rearrangewords *)

BEGIN   (* main action block *)
   writeln(' Piglatin generator');
   writeln;
   writeln(' Type ''end'' when finished');
   writeln;
   flag := true;
   REPEAT   (* process a line of text *)
      initialize;
      readinput;
      IF (english [1] = 'e') AND (english [2] = 'n') AND
         (english [3] = 'd') THEN flag := false;
      countwords;
      rearrangewords
   UNTIL flag = false
END.
```

لاحظ أن هذا البرنامج يحتوى على تحديد محتويات إحدى المنظومات لمنظومة أخرى ( فى إجراء initialize ) وتحديد عناصر فردية لمنظومة ( فى readinput , rearrangewords ) ..لاحظ أيضا أن محتوى متغير سلسلة ( أى محتوى سلسلة مضغوطة ) يكتب بعبارة writeln واحدة فى الإجراء rearrangewords . ويجب على عملية الإدخال المناظرة بالطبع أن تؤدى عنصرا عنصرا كما فى الإجراء readinput .

دعنا نعتبر الآن ماذا يحدث عندما ينفذ البرنامج . فيما يلى حوارًا نمطيًا ، موضوع خط فيه تحت استجابة المستفيد .

```
Piglatin generator

Type 'end' when finished

Pascal is a popular structured programming language
ascalPa sia aa opularpa tructuredsa rogrammingpa anguagela

baseball is the great American pastime,
aseballba sia heta reatga mericanAa astime,pa

though there are many who prefer football
houghta hereta reaa anyma howa referpa ootballfa

please do not sneeze in the computer room
leasepa oda otna neezesa nia heta omputerca oomra

end
ndea
```

لاحظ أن البرنامج لايحتوى على أى إشارة فى النص لعلامات تنقيط ، أو حروف كبيرة ، أو أصوات مزدوجة الحرف ( مثل th أو sh وخلافه ) . وهذه الأمور متروكة ليجريها القارئ كتمارين .

## ٦ – مؤشرات منظومة متغيرة الطول : 6. VARIABLE-LENGTH ARRAY PARAMETERS

لقد رأينا بالفعل أنه يمكن تمرير كل محتويات المنظومة إلى إجراء ، أو إلى دالة ، على أن تكون المؤشرات الفعلية والمؤشرات الرسمية المناظرة لها من نفس النوع الصريح ، وتحتوى على ذلك على نفس عدد العناصر ( انظر قسم ٣ من هذا الفصل وبصفة خاصة مثال ٩ –١٦) . وفى بعض التطبيقات قد يكون مطلوبا وجود ( داخل إجراء أو دالة ) مؤشر رسمى لايكون محدد له حد أقصى لعدد عناصره بصورة مطلقة . وهذه السمة تسمح بالاتصال بالإجراء أو بالدالة من عدة مواقع بمؤشرات فعلية مختلفة الحجم عند كل نقطة من نقاط الاتصال .

يمكن استخدام المؤشرات الرسمية ذات الطول المتغير للمنظومة داخل إجراء أو دالة ، على أن توضح هذه المؤشرات بطريقة خاصة . وتعرف مثل هذه المؤشرات بأنها مؤشرات فعلية لمنظومة متطابقة .

ويوسع استخدام المؤشرات المتطابقة لمنظومة من تعميم برنامج البسكال . ومن ناحية أخرى ... فإن المتطلبات التكوينية المصاحبة لاستخدامها معقدة . وقد يختار المبرمجون المبتدئون على ذلك تجنب استخدامها حتى يصبحوا أكثر اعتيادا على صيغ مؤشرات المرور الأخرى .

دعنا نعتبر أولا مؤشرات منظومة متطابقة ذات بعد واحد . وطريقة تعريفها تعتمد على ما اذا كانت هذه المؤشرات هى مؤشرات قيمة ، أو مؤشرات متغير ، كما يتضح ذلك من المثال التالى .

مثال (٩–٢٨)

افرض أن sample هو إجراء يستخدم مؤشرًا متغيرًا رسميًا لمنظومة تسمى list من النوع الحقيقى . ويمكن على ذلك كتابة عنوان الإجراء كما يلى :

```
PROCEDURE sample (VAR list : ARRAY [first..last : integer] OF real);
```

وعلى هذا فإننا نرى أن list عبارة عن منظومة حقيقية النوع ، لها بعد واحد ، وقيم فهرسها هي أرقام صحيحة تتراوح من first إلى last . وتتحدد القيم العددية لكل من last , first بواسطة المؤشر الفعلى المناظر ، والذى يجب أن يكون منظومة حقيقية النوع ذات بعد واحد ، لها فهرس صحيح النوع .

اعتبر على سبيل المثال منظومة من النوع الحقيقى ، لها بعد واحد ، اسمها item ، ويظهر توضيحها فى المجموعة الرئيسية للبرنامج ، أى :

```
VAR item : ARRAY [1..100] OF real;
```

افرض الآن أن item هو مؤشر فعلى يمر إلى sample ، أى :

```
sample(item);
```

وعلى هذا ... يأخذ first القيمة 1 ، ويأخذ last القيمة 100 ، بحيث تعامل list كأنها منظومة بها 100 عنصر من النوع الحقيقى أثناء الاتصال بالإجراء .

فإذا ماعدلت عناصر list داخل الإجراء ، فإن عناصر الصفوفة المعدلة تعود إلى item كما هو الحال دائما مع متغير المؤشر الرسمى .

## مـثال (٩-٢٩)

فيما يلى تخطيط هيكليا لبرنامج بسكال ، يستخدم دالة تقوم بعد عدد الكلمات الموجودة فى السطر . وبهدف التوضيح ... دعنا نستخدم مؤشر قيمة متوافقة للمنظومة داخل الدالة .

```
PROGRAM dummy(input,output);
TYPE length = 1..80;
     words  = 1..20;
VAR heading : PACKED ARRAY [1..12] OF char;
    message : PACKED ARRAY [1..80] OF char;
    n1,n2   : words;

FUNCTION count (text : PACKED ARRAY [start..stop : length] OF char) : words;
BEGIN
   .
   .
   .
   count := . . .
END;

BEGIN (* main action statements *)
   .
   .
   n1 := count(heading);
   .
   .
   n2 := count(message);
   .
   .
END.
```

يحتوى المثال على معالم متعددة تتطلب بعض التوضيح . لاحظ أولا text تعتبر منظومة مضغوطة ، بها 12 عنصرا ، وذلك أثناء الإشارة الأولى إلى الدالة ، كما أن text تعتبر منظومة مضغوطة بها 80 عنصرا ، وذلك أثناء الإشارة الثانية إلى الدالة . وتعود الدالة فى كل حالة برقم صحيح ، تقع قيمته بين 1 , 20 . ( وهذا بافتراض أن كل سطر من أسطر النص يحتوى على كلمة واحدة على الأقل ، ولايحتوى على أكثر من 20 كلمة ) .

يلى ذلك ان المنظومتين المضغوطتين message , heading ، والمستخدمتين كمؤشرين فعليين فى الإشارة إلى الدالة هما متغيران سلسلة صحيحان . ( نفترض أن السلاسل ممثلة بواسطة هذين المتغيرين تقرأ داخل الكمبيوتر ، وإلا فتحدد قيمها قبل الإشارة إلى الدالة ) . وعلى أية حال ... فإن مؤشر القيمة الرسمى text ( أى مؤشر قيمة متوافقة المنظومة ) لايعتبر متغير سلسلة . والخواص الخاصة بمتغيرات السلسلة التى سبق مناقشتها فى قسم ٥ من هذا الفصل لايمكن استخدامها داخل الدالة . وهذا القيد يسرى على المنظومات المضغوطة من النوع الحرفى المستخدمة كمؤشرات منظومة متوافقة .

يمكن أن يكون لمؤشرات منظومة متوافقة أبعاد متعددة أو بعد واحد . وبصفة عامة ... تطبق نفس القاعدة ، بالرغم من وجود قيد واحد خاص باستخدام الضغط . وبالتحديد فإن البعد الداخلى جدا ( أى آخر بعد من الداخل ) يمكن ضغطه . فإذا ماأستخدم الضغط بهذه الطريقة ، فيجب تعريف مؤشر المنظومة المتوافقة بأنه منظومة من المنظومات المضغوطة ، كما هو موضح أدناه .

## مثـال (٩-٣٠)

اعتبر عنوان الإجراء التالى ، والذى يستخدم مؤشر منظومة متوافقة ذات بعدين :

```
PROCEDURE sample (text : ARRAY [first..last : integer]
                  OF PACKED ARRAY [c1..cn : integer] OF char);
```

لاحظ أن text هو مؤشر قيمة منظومة متوافقة ، وعناصره هى حروف فردية .

إذا مانظر على ذلك لصفحة من النص page of text ، حيث أول فهرس يمثل رقم السطر ، وثانى فهرس يمثل موقع الحرف داخل سطر معين . لاحظ أيضا أن كل من الفهرسين من النوع الصحيح ، بالرغم من أن عناصر المنظومة نفسها من النوع الحرفى .

وهناك قيود معينة تجب مراعاتها عند استخدام مؤشرات منظومة متوافقة . لقد ناقشنا بالفعل اثنين من هذه القيود ، وهما أنه لايمكن معاملة منظومة مضغوطة من النوع الحرفى كمتغير سلسلة داخل الإجراء أو الدالة التى تمر إليها ( أنظر مثال ٩ -٢٩) . ويمكن استخدام الضغط مع البعد الداخلى جدا فقط بالنسبة للمنظومة متعددة الأبعاد . وهناك قيد إضافى خاص بمرور مؤشر منظومة متوافقة إلى إجراء آخر ، أو إلى دالة أخرى . فى مثل هذه الحالات يجب تعريف المؤشر الرسمى الموجود داخل الجزء الثانوى بأنه مؤشر متغير منظومة متوافقة ، بدلا من أنه مؤشر قيمة منظـومة متوافقة .

## مثال (٩-٣١)

فيما يلى تخطيطا هيكليا لبرنامج بسكال

```
PROGRAM nothing(input,output);
VAR item : ARRAY [1..100] OF real;

FUNCTION largest (VAR numbers :ARRAY [start..stop : integer] OF real) : real;
BEGIN
   .
   .
   largest := . . .
END;

PROCEDURE process (VAR list : ARRAY [first..last : integer] OF real);
VAR count : real;
BEGIN
   .
   .
   count := largest(list);
   .
   .
END;

BEGIN (* main block *)
   .
   .
   process(item);
   .
   .
END.
```

تمر فى هذا المثال المنظومة الحقيقية التى بها 100 عنصرا ، والمسماه item إلى الإجراء process . والمؤشر الرسمى المناظر هو مؤشر متغير منظومة متوافقة يسمى list . وتمر هذه المنظومة من خلال process إلى الدالة largest كما أن المؤشر الرسمى المناظر فى largest هو أيضا مؤشر متغير لمنظومة متوافقة يسمى numbers .

لاحظ أن مؤشر المنظومة المتوافقة الموجود فى process ( وهو list ) يصبح فيما بعد مؤشرا فعليا عندما يشار إلى الدالة largest . وعلى هذا ... يجب تعريف numbers ( المؤشر الرسمى داخل largest ) كمؤشر متغير منظومة متوافقة ، بدلا من أنه مؤشر قيمة منظومة متوافقة .

وهناك قيد آخر يجب ملاحظته عند تعريف مؤشرى منظومة متوافقة معا . فى مثل هذه الحالة يجب أن تكون المؤشرات الفعلية المناظرة فى أى إشارة فردية من نفس النوع . وهذا القيد موضح فى المثال التالى .

## مثال (٩-٣٢)

جمع جدولين من الأعداد ، دعنا نعتبر مرة أخرى برنامج جمع جدولين من الأعداد ، والذى سبق ذكره فى مثال ٩ -١١. دعنا – على أية حال – نستخدم الآن مؤشرات منظومة متوافقة فى تمثيل الجدولين داخل أجزاء البرنامج .

```
PROGRAM table3(input,output);

(* THIS PROGRAM READS IN TWO TABLES OF INTEGERS,
   CALCULATES THE SUMS OF THEIR RESPECTIVE ELEMENTS,
   AND WRITES OUT THE RESULTING TABLE CONTAINING THE SUMS *)

(* THE PROGRAM MAKES USE OF CONFORMANT ARRAY PARAMETERS *)

VAR  a,b,c : ARRAY [1..20,1..30] OF integer;
     row,nrows : 1..20;
     col,ncols : 1..30;

PROCEDURE readinput (VAR t : ARRAY [firstrow..lastrow : integer;
                                    firstcol..lastcol : integer] OF integer);

(* Read the elements of one table *)


BEGIN
   FOR row := 1 TO nrows DO
      BEGIN
         writeln(' Enter data for row no. ',row:2);
         FOR col := 1 TO ncols DO read(t[row,col]);
         writeln
      END
END;   (* readinput *)

PROCEDURE computesums (VAR t1,t2,t3 : ARRAY [firstrow..lastrow : integer;
                                             firstcol..lastcol : integer] OF integer)

(* Sum the elements of two similar tables *)


BEGIN
FOR row := 1 TO nrows DO
    FOR col := 1 TO ncols DO
        t3[row,col] := t1[row,col] + t2[row,col]
END;   (* computesums *)

PROCEDURE writeoutput (t : ARRAY [firstrow..lastrow : integer;
                                  firstcol..lastcol : integer] OF integer);

(* Write out the elements of a table *)


BEGIN
   writeln(' Sums of the elements:');
   writeln;
   FOR row := 1 TO nrows DO
      BEGIN
         FOR col := 1 TO ncols DO write(t[row,col]:4);
         writeln
      END
END;   (* writeoutput *)
```

(تكملة البرنامج فى الصفحة التالية)

```
BEGIN   (* main action block *)
   write(' How many rows? (1..20) ');
   readln(nrows);
   writeln;
   write(' How many columns? (1..30) ');
   readln(ncols);
   writeln;
   writeln(' First table:');
   writeln;
   readinput(a);
   writeln(' Second table:');
   writeln;
   readinput(b);
   computesums(a,b,c);
   writeoutput(c)
END.
```

ومن المفيد مقارنة صيغة هذا البرنامج من البرنامج الموجود فى مثال ٩ - ١٦. لاحظ على سبيل المثال أن تعريف النوع غير موجود فى الصيغة الحالية . بدلا من ذلك توجد مواصفات المنظومة داخل التوضيحات المتغيرات .

ومن المهم جدا معرفة حقيقة أن كل الإجراءات تستخدم مؤشرات منظومة متوافقة كمؤشرات رسمية . ويستخدم إجراءان ، وهما readinput , computesums مؤشرات متغير منظومة متوافقة ، بينما يستخدم الإجراء الثالث وهو writeoutput مؤشر قيمة منظومة متوافقة . لاحظ أن الثلاثة مؤشرات منظومة متوافقة t3 , t2 , t1 معرفة معا فى الإجراء computesums . لاحظ أيضا أن الإشارة المناظرة إلى computesums داخل المجموعة الرئيسية تحتوى على ثلاثة مؤشرات فعلية ، وهى c , b , a ، وكلها من نفس النوع كما هو مطلوب .

وقبل ترك هذا القسم ، فإننا نذكر القارئ مرة أخرى بأن استخدام مؤشرات منظومة متوافقة معقد بعض الشئ وقد لايكون مناسبا على أية حال مع المبرمجين المبتدئين . وأكثر من ذلك ... فإن بعض صيغ البسكال لاتستخدم مؤشرات منظومة متوافقة . وهذا صحيح بصفة خاصة مع أجهزة الميكروكمبيوتر . وحتى إذا ماكانت هذه السمة متاحة ، فيجب أن يتم التدريب عليها جيدا قبل البدء فى استخدامها فى البرامج .

## أسئلة للمراجعة : Review Questions

(١) ماهى الأربعة أنواع المختلفة للبيانات المرتبة ؟

(٢) ماهى الخواص الخاصة بمنظومة كبيانات مرتبة النوع ؟

(٣) اقترح طريقة عملية لرؤية منظومة ذات بعد واحد .

(٤) مامعنى عنصر المنظومة ؟ ومانوع البيانات التى يمكن استخدامها كعناصر فى منظومة ؟

(٥) مامعنى فهرس ( أو دليل ) ؟ مانوع البيانات التى يمكن استخدامها كفهرس ؟ ( قارن إجابتك مع إجابتك للسؤال السابق ) .

(٦) كيف يمكن الاتصال بعنصر واحد من عناصر المنظومة ؟

(٧) وضح كيف يمكن تعريف منظومة داخل جزء توضيح المتغيرات فى البرنامج .

(٨) لخص قواعد استخدام عنصر منظومة داخل تعبير ، أو داخل عبارة تحديد ، أو داخل عبارة read أو write .. الخ . كيف يمكن كتابة الفهارس ؟ وماهى القيود الموضوعة على الطريقة التى تكتب بها الفهارس ؟ .

(٩) هل يمكن استخدام عناصر البيانات المتعددة كعناصر منظومة ؟

(١٠) وضح كيف يمكن استخدام منظومة فى تمثيل سلسلة .

(١١) اقترح طريقة عملية لرؤية منظومة ذات بعدين .

(١٢) لخص قواعد تعريف منظومة ذات أبعاد متعددة كجزء من توضيح المتغيرات . هل يجب أن تكون كل الفهارس من نفس نوع البيانات ؟

(١٣) لخص قواعد الإشارة إلى عنصر من عناصر منظومة متعددة الأبعاد .

(١٤) وضح سبب الحاجة إلى دورات متداخلة لبعض تطبيقات المنظومات متعددة الأبعاد . ماهو الغرض من كل دورة ؟

(١٥) صف كيف يمكن التعبير عن منظومة ذات بعدين بأنها منظومة ذات بعد واحد ، وكل عنصر من عناصرها الفردية عبارة عن منظومة ذات بعد واحد . هل يمكن توسيع هذا المفهوم ليشمل منظومات لها أبعاد أكثر من ذلك ؟

(١٦) صف كيف يمكن توضيح عدة منظومات من نفس النوع عن طريق تعريف نوع المنظومة ، ثم توضيح أن المنظومات الفردية عبارة عن متغيرات من نفس النوع . هل يجب على كل المنظومات الموضحة بهذه الطريقة أن يكون لها نفس المدى ؟

(١٧) كيف يمكن تحديد جميع عناصر منظومة لمنظومة أخرى باستخدام عبارة تحديد واحدة ؟ ماهى القيود الموضوعة على هاتين المنظومتين ؟

(١٨) هل يمكن أن تظهر جميع محتويات المنظومة فى تعبير عددى أو تعبير بوليان ؟

(١٩) هل يمكن قراءة جميع محتويات المنظومة داخل الكمبيوتر باستخدام عبارة read أو readln واحدة ؟

(٢٠) هل يمكن كتابة جميع محتويات المنظومة خارج الكمبيوتر باستخدام عبارة write أو writeln واحدة ؟

(٢١) ماهى القيود الموضوعة على مرور جميع محتويات المنظومة إلى أجزاء أو إلى دالة ؟

(٢٢) صف طريقة مقنعة لمرور جميع محتويات منظومة إلى إجراء أو إلى دالة .

(٢٣) هل يمكن مرور عنصر منظومة واحد إلى إجراء أو إلى دالة ؟

(٢٤) ماهى المنظومة المضغوطة ؟ وكيف تختلف المنظومات المضغوطة عن المنظومات المعتادة ؟

(٢٥) ماهى الميزات والعيوب الممكنة التى تصاحب استخدام المنظومات المضغوطة ؟

(٢٦) لأى نوع من أنواع التطبيقات تكون المنظومات المضغوطة ذات ميزة خاصة ؟

(٢٧) هل يمكن للمنظومة المضغوطة أن يفك ضغطها ؟ وهل يمكن للمنظومة غير المضغوطة أن تضغط ؟ وضح ذلك .

(٢٨) لخص القواعد المصاحبة لاستخدام الإجرائين القياسيين pack , unpack .

(٢٩) هل يمكن تمرير جميع محتويات منظومة مضغوطة إلى إجراء أو إلى دالة ؟

(٣٠) هل يمكن تمرير عنصر واحد من عناصر منظومة مضغوطة إلى إجراء أو إلى دالة ؟

(٣١) ماهو متغير السلسلة ؟ وماهى العلاقة بين المنظومات المضغوطة ومتغيرات السلاسل ؟

(٣٢) صف العمليات الخاصة بالمنظومة المضغوطة التى يمكن استخدامها مع سلاسل أو مع متغيرات سلاسل فقط .

(٣٣) كيف تفسر المؤثرات العلاقية عند استخدامها فى وصل سلاسل أو متغيرات سلاسل لتكوين تعبير بوليان ؟ وبصفة خاصة كيف تفسر المؤثرات > و = > و < و = < ؟

(٣٤) ماهو مؤشر المنظومة المتوافقة ؟ وفى أى غرض يستخدم ؟

(٣٥) لخص قواعد تعريف مؤشرات منظومة متوافقة ذات بعد واحد . ميز بين مؤشرات قيمة منظومة متوافقة ، ومؤشرات متغير منظومة متوافقة .

(٣٦) لخص قواعد تعريف مؤشر منظومة متوافقة ذات أبعاد متعددة . قارن ذلك بالقواعد المذكورة فى إجابة السؤال السابق .

(٣٧) هل يمكن استخدام الضغط مع منظومة متعددة الأبعاد ؟ وضح ذلك بالتفصيل .

(٣٨) افرض أن إحدى المنظومات المضغوطة من النوع الحرفى تم تمريرها إلى إجراء أو إلى دالة كمؤشر منظومة متوافقة . هل يمكن معاملة هذه المنظومة كمتغير سلسلة داخل الإجراء أو داخل الدالة ؟

(٣٩) افرض أن مؤشر رسمى لمنظومة متوافقة داخل إجراء معين سوف يمرر إلى إجراء آخر . ( أى أن المؤشر الرسمى لمنظومة متوافقة سيصبح مؤشرا فعليا ليمر إلى إجراء ثانوى ) . بأى طريقة يجب تعريف مؤشر منظومة متوافقة داخل الإجراء الثانوى ؟

(٤٠) افرض أنه تم تعريف اثنين أو أكثر من مؤشرات رسمية لمنظومة متوافقة معا داخل عنوان أحد الإجراءات . ماهو الشرط الذى يجب تحقيقه بالمؤشرات الفعلية لمنظومة متوافقة تناظر هذه المؤشرات ؟

(٤١) هل مؤشرات المنظومة المتوافقة موجودة فى كل صيغ البسكال ؟

## Solved Problems
## مسائل محلولة :

(٤٢) فيما يلى عدة توضيحات صحيحة لمنظومات :

```
(a)  VAR values : ARRAY [0..100] OF real;
(b)  VAR list : ARRAY [-10..10] OF real;
(c)  TYPE days = (sun,mon,tues,wed,thurs,fri,sat);
     VAR calories : ARRAY [days] OF integer;
(d)  TYPE days = (sun,mon,tues,wed,thurs,fri,sat);
     VAR sunshine : ARRAY [1..365] OF days;

(e)  TYPE color = (blue,brown,gray,green,black);
          size  = (small,medium,large);
     VAR  coats  : ARRAY [size] OF color;
(f)  TYPE color = (blue,brown,gray,green,black);
          size  = (small,medium,large);
     VAR  coats  : ARRAY [size,color] OF integer;
(g)  TYPE color = (blue,brown,gray,green,black);
          size  = (small,medium,large);
     VAR  coats  : ARRAY [size] OF ARRAY [color] OF integer;
(h)  VAR parts : ARRAY ['A'..'Z', 1..10000] OF integer;
(i)  CONST max1  = 2000;
           max2  = 5000;
     TYPE range1 = 1..max1;
          range2 = 1..max2;
     VAR  tally  : ARRAY [range1,range2] OF char;
(j)  VAR symbol : ARRAY [1..12, 1..20, 0..5] OF boolean;
(k)  TYPE days = (sun,mon,tues,wed,thurs,fri,sat);
     VAR sunshine : PACKED ARRAY [1..365] OF days;
(l)  VAR name : PACKED ARRAY [1..80] OF char;
(m)  VAR page : ARRAY [1..66] OF PACKED ARRAY [1..80] OF char;
```

(٤٣) التخطيطات الهيكلية التالية توضح استخدام منظومات وعناصر منظومات فى مواقف برمجة .

```
(a)  PROGRAM sample(input,output);
     VAR square : ARRAY [1..100] OF integer;
         count  : 1..100;
     BEGIN
        .
        .
        FOR count := 1 TO 100 DO square [count] := sqr(count);
        .
        .
     END.

(b)  PROGRAM sample(input,output);
     VAR table : ARRAY [1..20, 1..8] OF real;
           i,m : 1..20;
           j,n : 1..8;
     BEGIN
```

```
   .
   .
   readln(m,n);
   FOR i := 1 TO m DO
      BEGIN
         FOR j := 1 TO n DO read( table [i,j]);
         writeln
      END;
   .
   .
   FOR i := 1 TO m DO
      BEGIN
         FOR j := 1 TO n DO write( table [i,j]);
         writeln
      END
END
```

(*c*)

```
PROGRAM sample(input,output);
VAR value : ARRAY [0..100] OF real;
    index,max : 0..100;
BEGIN
   .
   .
   readln(max);
   .
   .
   value[0] := 0;
   FOR index := max DOWNTO 0 DO
      BEGIN
         value[index] := index MOD 8;
         IF value[index] < 5.0 THEN value[0] := value[0] + value[index]
      END;
   .
   .
   END.
```

(*d*)

```
PROGRAM sample(input,output);
TYPE color = (red,white,blue,yellow,green);
VAR foreground : PACKED ARRAY [1..5] OF color;
    background : ARRAY [1..5] OF color;
    index : 1..5;
BEGIN
   .
   .
   FOR index := 1 TO 5 DO background[index] := . . .;
   .
   .
   pack(background,1,foreground);
   .
   .
END.
```

(*e*)

```
PROGRAM sample(input,output);
VAR city,office : PACKED ARRAY [1..3] OF char;
    order : 1..maxint;
BEGIN
```

```
        .
        .
        city := 'CHI';
        .
        .
        readln(order);
        IF order < 100 THEN office := city
                       ELSE IF order < 1000 THEN office := 'NYC'
                                            ELSE office := 'ATL';
        .
        .
        writeln(office);
        .
        .
     END.
```

لاحظ أن كلاً من city , office هو متغير سلسلة .

(*f*)

```
PROGRAM sample(input,output);
VAR list1 : ARRAY [1..40] OF integer;
    list2 : ARRAY [1..80] OF integer;

   PROCEDURE sort(VAR items : ARRAY [first..last : integer] OF integer);
   VAR . . .;   (* local variables *)
   BEGIN
      .
      .
      (* process the elements of items *)
      .
      .
   END;
BEGIN   (* main action statements *)
   .
   .
   (* read elements of list1 *)
   .
   .
   sort(list1);
   .
   .
   (* read elements of list2 *)
   .
   .
   sort(list2);
   .
   .
END.
```

لاحظ أن كلاً من list1 , list2 منظومة مختلفة فى حجمها .

## مشاكل متكاملة : Supplementary Problems

(٤٤) التخطيط الهيكلى التالى يوضح مواقف مختلفة عديدة تشمل استخدام مصفوفات وعناصر مصفوفات ، بعضها مكتوب بطريقة خاطئة . عرف كل الأخطاء .

(*a*)
```
TYPE notes = (do,re,mi,fa,sol,la,ti);
VAR tune : ARRAY [1..256] OF notes;
```

(*b*)
```
TYPE notes = (do,re,mi,fa,sol,la,ti);
VAR count : ARRAY [notes] OF integer;
```

(*c*)
```
TYPE notes = (do,re,mi,fa,sol,la,ti);
TYPE chorus = PACKED ARRAY [1..256] OF notes;
VAR first,second,trio,coda : chorus;
```

(*d*)
```
VAR demo : ARRAY [12..-5, 0..48] OF real;
```

(*e*)
```
PROGRAM sample(input,output);
VAR list : ARRAY [1..100] OF real;
BEGIN
   .
   .
   list [2.0] := 5.5;
   .
   .
END.
```

(*f*)
```
PROGRAM sample(input,output);
VAR list : ARRAY [1..100] OF real;
    index : 1..100;
BEGIN
   .
   .
   FOR index := -100 TO 100 DO
      IF index < 0 THEN list [index] := 0
                   ELSE list [index] := index;
   .
   .
END.
```

(*g*)
```
PROGRAM sample(input,output);
VAR list : ARRAY [1..100] OF real;
    index : 1..100;
    max : real;
BEGIN
   .
   .
   max := 0;
   FOR index := 1 TO 100 DO
      IF list [index] > max THEN max := list [index];
   .
   .
END.
```

(*h*)
```
PROGRAM sample(input,output);
VAR item : ARRAY ['A'..'Z', 1..50] OF integer;
BEGIN
   .
   .
   item [7] := 283;
   .
   .
END.
```

(*i*)
```
PROGRAM sample(input,output);
VAR size : ARRAY [1..50] OF integer;
BEGIN
   .
   .
   size [12] := 41.5;
   .
   .
END.
```

(*j*)
```
PROGRAM sample(input,output);
VAR item : ARRAY ['A'..'Z', 1..50] OF integer;
BEGIN
   .
   .
   item [7,'M'] := 283;
   .
   .
END.
```

(*k*)
```
PROGRAM sample(input,output);
VAR item : ARRAY ['A'..'Z', 1..50] OF integer;
BEGIN
   .
   .
   item ['C',9] := 60;
   .
   .
END.
```

(*l*)
```
PROGRAM sample(input,output);
TYPE list = ARRAY [1..200] OF real;
VAR sales, costs, profits : list;
BEGIN
   .
   .
   profits := sales - costs;
   .
   .
END.
```

(*m*)
```
PROGRAM sample(input,output);
VAR name : PACKED ARRAY [1..5] OF char;
BEGIN
   .
   .
   IF name = 'Jones' THEN writeln(name);
   .
   .
END.
```

(*n*)
```
PROGRAM sample(input,output);
VAR table : ARRAY [1..20, 1..80] OF integer;
BEGIN
   .
   .
   read(table);
   .
   .
   write(table);
   .
   .
END.
```

(*o*)
```
PROGRAM sample(input,output);
VAR table : ARRAY [1..1000, 1..66, 1..132] OF integer;
    i,j,k : integer;
BEGIN
   .
   .
   FOR i := 1 TO 1000 DO
      FOR j := 1 TO 66 DO
         FOR k := 1 TO 132 DO
            BEGIN
               table [i,j,k] := i + j + k - 2;
               writeln(i:4, j:2, k:3, table[i,j,k]:4)
            END;
   .
   .
END.
```

(*p*)
```
PROGRAM sample(input,output);
VAR buffer : ARRAY [1..80] OF char;
    target : PACKED ARRAY [1..80] OF char;
BEGIN
   .
   .
   pack(buffer,target,5);
   .
   .
   unpack(target,12,buffer);
   .
   .
END.
```

(*q*)
```
PROGRAM sample(input,output);
VAR line : PACKED ARRAY [1..80] OF char;
    name : PACKED ARRAY [1..40] OF char;
 address : PACKED ARRAY [1..40] OF char;
BEGIN
   .
   .
   IF name <> 'end' THEN line := name + address;
   .
   .
END.
```

(*r*)
```
PROGRAM sample(input,output);
VAR foreground,background : PACKED ARRAY [1..5] OF char;
    flag : boolean;
BEGIN
   .
   .
   background := 'black';
   .
   .
   IF flag THEN foreground := background
           ELSE foreground := 'green';
   .
   .
END.
```

(٤٥) فيما يلي عدة تخطيطات هيكلية توضح استخدام منظومات أو عناصر منظومات مع إجراءات أو دوال ، بعضها مكتوب بطريقة خاطئة . عرف كل الأخطاء .

(*a*)
```
PROGRAM sample(input,output);
VAR list : ARRAY [1..100] OF integer;

   PROCEDURE process (dummy : ARRAY [1..100] OF integer);
   BEGIN
      .
      .
      .
   END;

   BEGIN   (* main action statements *)
      .
      .
      process(list);
      .
      .
   END.
```

(*b*)
```
PROGRAM sample(input,output);
TYPE list = ARRAY [1..100] OF real;
VAR list1,list2 : list;
   PROCEDURE module (VAR x : list);
   BEGIN
      .
      .
      (* process the elements of x *)
      .
   END;
BEGIN   (* main action statements *)
   .
   .
   module(list1);
   .
   .
   module(list2);
   .
   .
END.
```

(*c*)
```
PROGRAM sample(input,output);
VAR sales : ARRAY [1..100] OF integer;
   result : integer;
   FUNCTION funct1 (x : integer) : integer;
   BEGIN
      .
      .
      .
      funct1 := . . .
   END;
BEGIN   (* main action statements *)
   .
   .
   result := funct1 (sales [6] + sales [12]);
   .
   .
END.
```

(*d*)
```
PROGRAM sample(input,output);
VAR line : PACKED ARRAY [1..80] OF char;

   PROCEDURE proc1 (dummy : char);
   BEGIN
      .
      .
      .
   END;

   BEGIN   (* main action statements *)
      .
      .
      proc1(line[40]);
      .
      .
   END.
```

(*e*)
```
PROGRAM sample(input,output);
VAR list  : ARRAY [1..100] OF integer;
    items : ARRAY [1..25] OF integer;

  PROCEDURE process (x : ARRAY [first..last : integer] OF integer);
  BEGIN
     .
     .
     .
  END;

BEGIN   (* main action statements *)
   .
   .
   process(list);
   .
   .
   process(items);
   .
   .
END.
```

(*f*)
```
PROGRAM sample(input,output);
VAR stockno : ARRAY [1..100] OF integer;
       cost : ARRAY [1..100] OF real;

  PROCEDURE process(x : ARRAY [first..last : integer] OF integer);
  BEGIN
     .
     .
     .
  END;

BEGIN   (* main action statements *)
   .
   .
   process(stockno);
   .
   .
   process(cost);
   .
   .
END.
```

(*g*)
```
PROGRAM sample(input,output);
VAR page : PACKED ARRAY [1..66, 1..80] OF char;

  FUNCTION count (text : PACKED ARRAY
     [first..last : integer; start..stop : integer]OF char) : integer;
  BEGIN
     .
     .
     .
     count := . . .
  END;
```

```
    BEGIN   (* main action statements *)
      .
      .
      (* read the elements of text *)
      .
      .
      writeln(count(text));
      .
      .
    END.
```

(*h*)
```
PROGRAM sample(input,output);
VAR name : PACKED ARRAY [1..80] OF char;

   PROCEDURE proc1 (line : PACKED ARRAY [first..last : integer] OF char);
   BEGIN
      .
      .
      writeln(line);
      .
      .
   END;

BEGIN   (* main action statements *)
   .
   .
   (* read the elements of name *)
   .
   .
   proc1(name);
   .
   .
END.
```

(*i*)
```
PROGRAM sample(input,output);
VAR item : ARRAY [1..100] OF real;

   PROCEDURE process (numbers : ARRAY [start..stop : integer] OF real);
   BEGIN
      .
      .
      .
   END;
   PROCEDURE module (x : ARRAY [start..stop :integer] OF real);
   BEGIN
      .
      .
      process(x);
      .
      .
   END;

BEGIN   (* main action statements *)
   .
   .
```

```
      module(item);
      .
      .
      process(item);
      .
      .
   END.
```

(*j*)
```
   PROGRAM sample(input,output);
   VAR a : ARRAY [1..20] OF real;
       b : ARRAY [1..50] OF integer;

      PROCEDURE. demo (VAR x,y : ARRAY [first..last : integer] OF real);
      BEGIN
         .
         .
         .
      END;

   BEGIN   (* main action statements *)
      .
      .
      demo(a,b);
      .
      .
   END.
```

## مشاكل برمجة : Programming Problems

(٤٦) عدل البرنامج المعطى فى مثال ٩ – ٨ ، بحيث يعاد ترتيب الأعداد فى تسلسل جبرى منخفض القيم ( أى من الأكبر إلى الأصغر ) . اختبر البرنامج مستخدما البيانات المعطاه فى مثال ٩ – ٨

(٤٧) عدل البرنامج المعطى فى مثال ٩ – ٨ بحيث يمكن تنفيذ إعادة ترتيب أى مما يلى :

(أ) من الأصغر إلى الأكبر بالنسبة للقيمة .

(ب) من الأصغر إلى الأكبر جبريا .

(ج) من الأكبر إلى الأصغر بالنسبة للقيمة .

(د) من الأكبر إلى الأصغر جبريا .

مع تواجد قائمة تسمح للمستفيد أن يختار أى إعادة ترتيب يريد استخدامها فى كل مرة يقوم بتنفيذ البرنامج . اختبر البرنامج مستخدما القيم العشر التالية .

| | |
|---|---|
| 4.7 | −8.0 |
| −2.3 | 11.4 |
| 12.9 | 5.1 |
| 8.8 | −0.2 |
| 6.0 | −14.7 |

(٤٨) عدل البرنامج المعطى فى مثال ٩ -١٦ لحساب الفروق ، بدلا من المجموع للعناصر المتناظرة فى كل من الجدولين . اختبر البرنامج مستخدما البيانات المعطاه فى مثال ٩ -١٦.

(٤٩) عدل البرنامج المعطى فى مثال ٩ – ٧ بحيث إنه يستخدم منظومات مضغوطة ، ومؤشرات منظومة متغيرة الطول . اختبر البرنامج مستخدما اسمك وعنوانك .

(٥٠) عدل برنامج pig latin الموجود فى مثال ٩ – ٢٧، بحيث يمكن احتواء علامات التنقيط ، والحروف الكبيرة ، والأصوات مزدوجة الحروف .

(٥١) اكتب برنامجا بلغة البسكال ، يدخل سطرا من أحد النصوص ، ويخزنه فى منظومة مضغوطة ، ثم يكتبها بعد ذلك . اجعل طول السطر غير محدد ( ويفصل بعودة العربة ) ، مفترضا أنها لن تزيد عن 80 خانة . اختبر البرنامج مستخدما أى سطر تعده بنفسك . قارن البرنامج ببرنامج مثال ٧ -٢٤ الذى يستخدم إجراء إعادة ، بدلا من استخدام منظومة . حدد أى طريقة هى الأفضل .

(٥٢) فيما يلى مجموعة مشاكل برمجة تستخدم درجات الطلبة التالية : ( الدرجات لستة امتحانات أداها كل طالب فى مقرر لغة البسكال ) .

| Name | Exam scores, percent | | | | | |
|---|---|---|---|---|---|---|
| Adams | 45 | 80 | 80 | 95 | 55 | 75 |
| Brown | 60 | 50 | 70 | 75 | 55 | 80 |
| Davis | 40 | 30 | 10 | 45 | 60 | 55 |
| Fisher | 0 | 5 | 5 | 0 | 10 | 5 |
| Hamilton | 90 | 85 | 100 | 95 | 90 | 90 |
| Jones | 95 | 90 | 80 | 95 | 85 | 80 |
| Ludwig | 35 | 50 | 55 | 65 | 45 | 70 |
| Osborne | 75 | 60 | 75 | 60 | 70 | 80 |
| Prince | 85 | 75 | 60 | 85 | 90 | 100 |
| Richards | 50 | 60 | 50 | 35 | 65 | 70 |
| Smith | 70 | 60 | 75 | 70 | 55 | 75 |
| Thomas | 10 | 25 | 35 | 20 | 30 | 10 |
| Wolfe | 25 | 40 | 65 | 75 | 85 | 95 |
| Zorba | 65 | 80 | 70 | 100 | 60 | 95 |

(أ) اكتب برنامجا تقليديا بلغة البسكال يقبل اسم كل طالب ودرجاته كمدخلات ، وحدد متوسط الدرجات لكل طالب ، ثم أخرج اسمه مع درجات كل الامتحانات والمتوسط المحسوب . ( ملاحظة : لاحظ أن كل اسم يتبعه فراغ ) . اجعل البرنامج عاما بقدر الإمكان ( انظر المشكلة رقم ٥٠ فى الفصل السادس الجزء k ) .

(ب) عدل البرنامج المكتوب للمشكلة السابقة ليسمح بترجيح غير متساو لدرجات الامتحانات . افرض بصفة خاصة أن كل امتحان من الامتحانات الأربعة الأولى يسهم بمقدار ١٥ ٪ من الدرجة النهائية ، وأن كل امتحان من الامتحانين الآخرين يمثل ٢٠ ٪ من الدرجة النهائية ( انظر المشكلة رقم ٥٠ فى الفصل السادس الجزء L ) .

(ج) وسع البرنامج المكتوب للمشكلة السابقة ، بحيث تحسب متوسط درجة الفصل ، بالإضافة إلى متوسطات الطلاب ( انظر المشكلة رقم ٥٠ فى الفصل السادس الجزء m ) .

(د) وسع البرنامج المكتوب للمشكلة السابقة ، بحيث يحسب انحراف متوسط درجة كل طالب من متوسط درجة الفصل . اكتب كمخرجات متوسط درجة الفصل ، يتبعها اسم كل طالب ، ودرجاته ، ومتوسطها ، والانحراف عن متوسط درجة الفصل . تأكد أن المخرجات مرتبة منطقيا ، ولها عناوين واضحة .

(٥٣) اكتب برنامجا بلغة البسكال ، ينتج جدولا يقيم المعادلة التالية :

$$y = 2e^{-0.1t} \sin 0.5t$$

حيث t تتغير من 0 إلى 60 . أسمح بزيادة تحدث فى t ، ويتم إدخالها كمؤشر مدخلات .

(٥٤) اكتب برنامجا بلغة البسكال ينتج جدولا لمعامل الفائدة المركبة (F/P) ، حيث :

$$A/P = \frac{i(1+i)^n}{(1+i)^n - 1}$$

تمثل F فى هذه العلاقة القيمة المستقبلية لمبلغ معين من المال . وتمثل P قيمته الحالية . كما تمثل i معدل الفائدة السنوية كنسبة مئوية . وتمثل n عدد السنوات . دع كل صف فى الجدول يناظر قيمة مختلفة من قيم n ، تتراوح قيم n من 1 إلى 30 ( أى أنه هناك 30 سطرا ) . ودع كل عمود يمثل معدل فائدة مختلف . الجدول يحتوى على معدلات فائدة : 4 , 4.5 , 5 , 5.5 , 6 , 6.5 , 7 , 7.5 , 8 , 8.5 , 9 , 9.5 , 10 , 11 , 12 , 15 فى المائة ( أى هناك 16 عمودا ) تأكد من وضع أسماء مناسبة للصفوف وللأعمدة .

(٥٥) اكتب برنامجا بلغة البسكال يعيد ترتيب قائمة بها كلمات ترتيبا أبجديا . لعمل ذلك أدخل الكلمات فى منظومة مضغوطة ذات بعدين من النوع الحرفى ، على أن يمثل كل صف كلمة كاملة . بعد ذلك يمكن إعادة ترتيب الصفوف بنفس الطريقة التى أعيد ترتيب قائمة بالأعداد بها من العدد الأصغر إلى العدد الأكبر ( انظر مثال٩ – ٨ ) .

استخدم البرنامج ليعيد ترتيب الأسماء التالية ، وكن حذرا عند استخدام الحروف الأولى من الأسماء .

| | | | |
|---|---|---|---|
| Washington | Polk | Arthur | Roosevelt, F. D. |
| Adams, J. | Taylor | Cleveland | Truman |
| Jefferson | Fillmore | Harrison, B. | Eisenhower |
| Madison | Pierce | McKinley | Kennedy |
| Monroe | Buchanan | Roosevelt, T. | Johnson, L. B. |
| Adams, J. Q. | Lincoln | Taft | Nixon |
| Jackson | Johnson, A. | Wilson | Ford |
| Van Buren | Grant | Harding | Carter |
| Harrison, W. H. | Hayes | Coolidge | Reagan |
| Tyler | Garfield | Hoover | |

(٥٦) اعتبر القائمة التالية التى تحتوى على أسماء بلاد وأسماء عواصم .

| | | | |
|---|---|---|---|
| Canada | Ottawa | Israel | Jerusalem |
| England | London | Italy | Rome |
| France | Paris | Japan | Tokyo |
| India | New Delhi | Mexico | Mexico City |

| | |
|---|---|
| People's Republic of China | Peking |
| United States | Washington |
| U.S.S.R. | Moscow |
| West Germany | Bonn |

اكتب برنامجا بلغة البسكال متداخلا ، يقبل اسم البلد كمدخلات ، ويكتب العاصمة المناظرة لها ، والعكس كذلك . صمم البرنامج ، بحيث يستمر حتى تظهر كلمة end كمدخلات .

(٥٧) أكتب برنامجا كاملا بلغة البسكال لكل مشكلة من المشاكل المذكورة أدناه . حدد نوع المنظومة الأكثر ملاءمة لكل مشكلة . تأكد من استخدام الأجزاء فى كل برنامج ، ومن كتابة عناوين واضحة للمخرجات ، ومن استخدام أنواع طبيعية للبيانات ومكونات تحكم ذات كفاءة .

(أ) افرض أنه معطى لنا جدول من الأعداد الصحيحة A ، به m صفا ، و n عمودا ، ولدينا قائمة بالأعداد الصحيحة X ، بها n عنصرا . ونريد إنتاج قائمة أعداد صحيحة جديدة Y ، يتم إعدادها عن طريق أداء العمليات التالية :

$$
\begin{aligned}
Y[1] &= A[1,1]*X[1] + A[1,2]*X[2] + \cdots + A[1,N]*X[N] \\
Y[2] &= A[2,1]*X[1] + A[2,2]*X[2] + \cdots + A[2,N]*X[N] \\
&\vdots \\
Y[M] &= A[M,1]*X[1] + A[M,2]*X[2] + \cdots + A[M,N]*X[N]
\end{aligned}
$$

اكتب كمخرجات بيانات المدخلات ( أى قيم العناصر X , A ) . يليها قيم عناصر Y .

استخدم البرنامج فى تشغيل البيانات التالية :

$$
A = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
6 & 7 & 8 & 9 & 10 & 11 & 12 & 13
\end{bmatrix}
$$

$$
X = \begin{bmatrix}
1 \\ -8 \\ 3 \\ -6 \\ 5 \\ -4 \\ 7 \\ -2
\end{bmatrix}
$$

(ب) افرض أن A جدول من الأعداد الحقيقية ، به K صفا ، وبه m عمودا ، وأن B جدول من الأعداد الحقيقية ، به m صفا ، وبه n عمودا . ونريد إنتاج جدولا جديدا C ، حيث يتحدد كل عنصر من عناصر الجدول C بالعلاقة التالية :

$$C[i,j] = A[i,1]*B[1,j] + A[i,2]*B[2,j] + \cdots + A[i,m]*B[m,j]$$

حيث :

$$i = 1, 2, \ldots, k$$
$$\text{and } j = 1, 2, \ldots, n.$$

اكتب عناصر الجداول الثلاثة C , B , A ، وتأكد من وجود عناوين واضحة لكل شئ .

استخدم البرنامج فى تشغيل مجموعة البيانات التالية :

$$A = \begin{bmatrix} 2 & -1/3 & 0 & 2/3 & 4 \\ 1/2 & 3/2 & 4 & -2 & 1 \\ 0 & 3 & -9/7 & 6/7 & 4/3 \end{bmatrix}$$

$$B = \begin{bmatrix} 6/5 & 0 & -2 & 1/3 \\ 5 & 7/2 & 3/4 & -3/2 \\ 0 & -1 & 1 & 0 \\ 9/2 & 3/7 & -3 & 3 \\ 4 & -1/2 & 0 & 3/4 \end{bmatrix}$$

(ج) اعتبر تسلسلاً من الأعداد الصحيحة $x_i$ ، حيث $i = 1, 2, \ldots, m.$ ويعرف المتوسط بأنه :

$$\bar{x} = \frac{(x_1 + x_2 + \cdots + x_m)}{m}$$

والانحراف عن المتوسط هو :

$$d_i = (x_i - \bar{x}), \qquad i = 1, 2, \ldots, m$$

والانحراف المعيارى هو :

$$s = \sqrt{\frac{(d_1^2 + d_2^2 + \cdots + d_m^2)}{m}}$$

أدخل أول m عنصرا من منظومة حقيقية ذات بعد واحد . احسب مجموع هذه العناصر ومتوسطها وانحرافها عن المتوسط والانحراف المعيارى لها ، وأكبر قيمة جبرية ، وأصغر قيمة جبرية . استخدم البرنامج فى تشغيل مجموعة البيانات التالية :

| | |
|---|---|
| 27.5 | 87.0 |
| 13.4 | 39.9 |
| 53.8 | 47.7 |
| 29.2 | 8.1 |
| 74.5 | 63.2 |

كرر الحسابات لعدد k من قوائم الأعداد المختلفة . احسب المتوسط الكلى ، والانحراف المعيارى الكلى ، وأقصى قيمة مطلقة ، وأقل قيمة مطلقة .

(د) افرض أنه لدينا مجموعة من القيم المجدولة لكل من x , y كما يلى :

$$\begin{matrix} y_0 & y_1 & y_2 & \cdots & y_n \\ x_0 & x_1 & x_2 & \cdots & x_n \end{matrix}$$

وأننا نريد الحصول على قيمة y التى تناظر x معينة ، وتقع بين قيمتين من قيم y الموجودة فى الجدول . وعادة ما تحل هذه المشكلة باستخدام الاستدلال interpolation ، أى بعمل كثيرة حدود y (x) تمر خلال n نقطة ، مثل :

$$y(x_0) = y_0,\ y(x_1) = y_1,\ .\ .\ .,\ y(x_n) = y_n$$

ثم تقويم y عند قيمة x المعطاه . وتستخدم صيغة لاجرانج Lagrange form لأداء هذا الاستدلال على كثيرة الحدود . ولعمل ذلك ... فإننا نكتب :

$$y(x) = f_0(x)*y_0 + f_1(x)*y_1 + \cdots + f_n(x)*y_n$$

حيث $f_i(x)$ هى كثيرة حدود ، بحيث إن :

$$f_i(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$$

لاحظ أن : $f_i(x_i) = 1$ ، $f_i(x_j) = 0$

حيث $x_j$ هى قيمة مجدولة من قيم x تختلف عن $x_i$ . وعلى هذا ... فإننا نتأكد من أن $y(x_i) = y_i$ .

اكتب برنامجا بلغة البسكال لقراءة n زوجا من البيانات ، حيث n لاتزيد عن 10 ، ثم الحصول على قيمة الاستدلال لـ y عند قيمة أو أكثر من قيم x . استخدم البرنامج فى الحصول على قيم y المحسوبة من الاستدلال عند قيم x=112 , x= 174 , x=13.7 من البيانات المذكورة أدناه . حدد عدد أزواج البيانات المجدولة المطلوبة فى كل حساب ، بغرض الحصول على قيمة لـ y ذات دقة معقولة .

(٥٨) اكتب برنامج بسكال متداخل يعد شفرة لسطر نصى أو يفك الشفرة .

لعمل شفرة للسطر النصى نفذ مايلى :

١ – حول كل رمز ، بما فى ذلك الفراغات ، الى مكافئة من شفرة ASCII .

٢ – انتج رقماً عشوائياً موجباً . أضف هذا الرقم الى مكافئ شفرة ASCII لكل رمز . ( يستخدم نفس الرقم العشوائى لبقية محتويات السطر النصى ) .

٣ – إفرض أن N1 يمثل أقل قيمة مسموح بها فى شفرة ASCII ، ويمثل N2 أعلى قيمة مسموح بها . فإذا كان الرقم الذى يتم الحصول عليه فى الخطوة رقم 2 ( أى مكافئ ASCII الأصلى مضافاً اليه الرقم العشوائى ) أكبر من N2 ويمثل دائماً ، على هذا ، أحد رموز ASCII .

٤ – أطبع الرموز المناظرة لقيم ASCII المشفرة .

عندما يراد فك الشفرة لسطر النص تعكس الخطوات . كن متأكداً على أية حال من استخدام نفس الرقم العشوائى فى فك الشفرة والذى سبق استخدامه فى عمل الشفرة .

# الفصل العاشر

# السـجلات

# Records

لقد درسنا فى الفصل السابق المنظومة ، وهى نوع من البيانات المرتبة التى يجب أن تكون كل عناصرها من نفس النوع . ونعيد انتباهنا الآن إلى نوع آخر مهم من أنواع البيانات المرتبة ، وهو السجل record الذى لاتحتاج العناصر المكونة له أن تكون من نفس النوع . وعلى هذا فيمكننا أن نشير إلى مجموعة من عناصر بيانات تختلف فى النوع عن بعضها . ويقال لعناصر البيانات الفردية أنها حقول fields داخل السجل .

## ١ – تعريف السجل : 1. DEFINING A RECORD

أسهل طريقة لتعريف السجل هى التعريف كجزء من توضيح المتغيرات ، وهذا يسمح لنا بتوضيح متغير فردى للسجل ، كما تعرف المتغيرات البسيطة ومتغيرات المنظومات .

والصيغة العامة لتوضيح متغير السجل هى .

VAR *record name* : RECORD *field 1*; *field 2*; . . .;*field n* END

حيث تمثل field 1 توضيح أول حقل ، وتمثل field 2 الحقل الثانى ، وهكذا . ويكتب توضيح كل حقل بطريقة تشبه توضيح المتغير الفردى ، أى على الصورة التالية :

*field name* : *type*

حيث field name هو معرف يمثل اسم الحقل ، و type هو نوع بيانات عنصر البيانات التى تشغل الحقل .

لاحظ أن الحقول الفردية داخل السجل مفصولة عن بعضها بواسطة فواصل منقوطة .

### مثـال(١٠-١)

افرض أن سجل العميل يحتوى على رقم العميل من النوع الصحيح ، وتحديد لحساب العميل من النوع الحرفى وموازنة العميل ، وهى عدد حقيقى . دعنا نمثل هذا السجل بمتغير سجل يسمى customer . وتوضيح المتغير هو مايلى :

```
VAR customer : RECORD
                  custno : integer;
                  custtype : char;
                  custbalance : real
               END;
```

لاحظ أن السجل يحتوى على حقل صحيح النوع اسمه custno ، وحقل من النوع الحرفى اسمه cust type ، وحقل من النوع الحقيقى اسمه Custbalance .

ولاتحتاج توضيحات الحقول الفردية أن تكتب على أسطر منفردة دائما كما فى هذا المثال . وعادة ماتكتب على أية حال بهذه الطريقة لتسهيل القراءة .

وهناك طريقة أخرى لتعريف السجل ، وعادة ماتكون أكثر نفعا من توضيح المتغير ، وهى تعريف نوع type السـجل . وبعد ذلك يمكن توضيح أن المتغيرات الفردية من هذا النوع . لاحظ التماثل مع الطرق المستخدمة فى تعريف المنظومات ، والتى سبق ذكرها فى الفصل السابق . وفى الصورة العامة يكتب تعريف نوع السجل على النحو التالى :

```
TYPE name = RECORD field 1; field 2; . . .;field n END
```

حيث إن معنى العناصر الفردية هو نفسه كما سبق ذكره .

## مثـال (١٠-٢)

اعتبر مرة أخرى سجل العميل المذكور فى المثال السابق . دعنا نعرف الآن نوع سجل اسمه account ، ومتغير مناظر اسمه customer كما سبق ذكره . يمكن تحقيق ذلك بكتابة

```
TYPE account = RECORD
                    custno : integer;
                    custtype : char;
                    custbalance : real
                 END;
VAR customer : account;
```

يجب أن يكون مفهوما أن هذا يحقق نفس الشئ ، مثل توضيح المتغير الموضح فى مثال ١٠- ١ . الطريقة الحالية عامة أكثر ، لأنها تسمح لنا بإدخال متغيرات سجلات إضافية إذا ماأردنا ذلك .

ويمكن أن يصاحب الحقول الفردية أنواع بيانات يحددها المستفيد ، وأنواع مدى جزئى ، وأنواع بيانات قياسية . أكثر من هذا ... يمكن ترتيب أنواع الحقول المدرجة Scalar . وعلى هذا ... يمكن لعنصر فردى أن يكون منظومة أو أى سجل آخر .

## مثـال (١٠-٣)

فيما يلى تعريف آخر للسجل الموضح فى مثال ١٠- ٢

```
TYPE status = (current,overdue,delinquent);
     account = RECORD
                    custno : 1..9999;
                    custtype : status;
                    custbalance : real
                 END;
VAR customer : account;
```

رقم العميل custno هنا عبارة عن عنصر بيانات من نوع المدى الجزئى ، ونوع العميل cust type هو عنصر بيانات من النوع المتعدد .

لاحظ أن عنصر البيانات من النوع المتعدد ( Status ) يجب أن يعرف بنفسه قبل أن يشار إليه فى توضيح السجل .

## مـثال (١٠-٤)

دعنا نضيف الآن اسم العميل إلى السجل الذى سبق ذكره . لعمل ذلك ... فإننا نعرف حقلا إضافيا يمثل منظومة مضغوطة من النوع الحرفى . وعلى هذا تصبح تعريف محتويات السجل على النحو التالى :

```
TYPE status = (current,overdue,delinquent);
     account = RECORD
                    custname : PACKED ARRAY [1..80] OF char;
                    custno : 1..9999;
                    custtype : status;
                    custbalance : real
               END;
VAR customer : account;
```

وفيما يلى تعريفًا آخر لهذا السجل الذى يعتبر عاما بعض الشئ .

```
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..80] OF char;
     account = RECORD
                    custname : line;
                    custno : 1..9999;
                    custtype : status;
                    custbalance : real
               END;
VAR customer : account;
```

يسمح هذا التعريف بتعريف معلومات من نوع السلسلة الأخرى بسهولة جدا ، وذلك داخل السجل الحالى ، أو فى أى جزء آخر من أجزاء البرنامج . وعلى هذا ... فإذا ماأردنا أن نضيف حقل عنوان إلى customer ، فيمكننا كتابة مايلى :

```
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..80] OF char;
     account = RECORD
                    custname : line;
                    custaddress : line;
                    custno : 1..9999;
                    custtype : status;
                    custbalance : real
               END;
VAR customer : account;
```

## مثـــال (١٠-٥)

دعنا نضيف الآن ( تاريخ ) آخر مبلغ دفعه العميل إلى سجل العميل . ويمكن أن يتحقق ذلك عن طريق تمثيل

التاريخ كسجل منفصل ، ثم اعتبار هذا السجل كحقل من حقول customer . وعلى هذا ... يصبح السجل الآن على النحو التالى :

```
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..80] OF char;
     date = RECORD
               month : 1..12;
               day : 1..31;
               year : 1900..2100
            END;
     account = RECORD
                  custname : line;
                  custaddress : line;
                  custno : 1..9999;
                  custtype : status;
                  custbalance : real;
                  lastpayment : date
               END;
VAR customer : account;
```

لاحظ أن السجل account يحتوى الآن على سجل آخر ، وهو lastpayment كأحد حقوله . كما نرى أيضا أن account يحتوى على منظومتين ( Custname , Custaddress ) ونوع مدى جزئى ( Custno ) ونوع بيانـــات متعـــددة ( Custtype ) ونوع بيانات قياسى ( Custbalance ) . وعلى هذا فإن هذا المثال يوضح المرونة الموجودة فى تعريف السجل .

ويجب أن يكون معروفا أيضا أنه يمكن للسجل أن يكون عنصر بيانات فرديًا فى نوع بيانات مرتب آخر مثل المنظومة . وعلى هذا يمكننا تعريف منظومات عناصرها سجلات وسجلات عناصرها ( أو بعض من عناصرها ) منظومات ، وما إلى ذلك .

**مثـال (١٠-٦)**

فيما يلى مثالا لمنظومة وعناصرها سجلات . وبصفة خاصة دعنا نعرف منظومة اسمها event ، وعناصرها الفردية هى data كما سبق تعريفها فى المثال السابق .

```
TYPE date = RECORD
               month : 1..12;
               day : 1..31;
               year : 1900..2100
            END;
VAR events : ARRAY [1..100] OF date;
```

وعلى هذا يكون لدينا منظومة ذات بعد واحد ( وهى list ) يمكن أن تحتوى على تواريخ dates مختلفة ، تصل إلى ١٠٠ تاريخ . لاحظ أن events معرف بأنه متغير ، وليس نوع بيانات .

وبعد تعريف نوع سجل فردى ، يمكننا توضيح متغيرات عديدة مختلفة كسجلات من هذا النوع . وهذه السمة موضحة فى المثال التالى .

## مــثال (١٠-٧)

اعتبر السجل account المعرف فى المثال رقم (١٠- ٥ ) . يمكننا الآن توضيح المتغيرين regular , preferred بأنهما سجلات من هذا النوع .

وعلى هذا نكتب مايلى :

```
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..80] OF char;
     date = RECORD
                month : 1..12;
                day : 1..31;
                year : 1900..2100
             END;
     account = RECORD
                    custname : line;
                    custaddress : line;
                    custno : 1..9999;
                    custtype : status;
                    custbalance : real;
                    lastpayment : date
                  END;
VAR preferred,regular : account;
```

ويمكن ضغط السجل كما هو الحال مع ضغط المنظومات تماما . ويسمح ذلك بتخزين السجل فى صورة أكثر إيجازا ، بالرغم من أن الاتصال بعنصر بيانات داخل مثل هذا السجل يستغرق وقتا أطول .

ويتحقق ضغط السجلات بكتابة PACKED RECORD ، بدلا من RECORD فى تعريف نوع السجل كما هو موضح فى المثال التالى .

## مثــال (١٠-٨)

دعنا نعيد تعريف نوع السجل account ، والذى ظهر فى المثال (١٠- ٢ ) لكى يكون سجلا مضغوطا .

```
TYPE account = PACKED RECORD
                    custno : integer;
                    custtype : char;
                    custbalance : real
                 END;
VAR customer : account;
```

وعلى هذا فإن المتغير Customer يمثل سجلا مضغوطا من نفس نوع السجل account .

ويشمل ضغط السجلات ، مثل ضغط المنظومات ، موازنة بين سرعة التنفيذ ومتطلبات الذاكرة . وعلى أية حال فإن هذه السمة ليست متسعة الانتشار فى استخدامها مثل سمة ضغط المنظومات . وكقاعدة عامة ... تستخدم هذه السمة فى التطبيقات التى تركز على نسخ محتويات السجلات فقط ، وذلك بدلا من التطبيقات الأكثر شيوعا ، والتى يعامل فيها عناصر فردية من عناصر السجلات .

وأخيرا يجب أن يذكر أن كل سجل يعتبر كينونة مستقلة بذاتها بالنسبة لتعريفات الحقول . وعلى هذا ... يجب أن يكون لكل حقل داخل أى سجل اسم فريد ، إلا أن نفس هذا الاسم الفريد فى السجل يمكن استخدامه فى سجلات أخرى . وفى كلمات أخرى يضيق مدى معرف الحقل ليشمل السجل المحدد المعرف داخله فقط .

مثال (١٠-٩)

فيما يلى تعريفا لنوعين مختلفين two و one .

```
TYPE one = RECORD
              a : real;
              b : integer;
              c : char
            END;
          = RECORD
              a : char;
              b,c : real
            END;
```

لاحظ أن اسماء الحقول الفردية c , b , a مكررة فى السجلين ، إلا أن أنواع البيانات المصاحبة لها مختلفة ، وهذا مسموح به ، حيث إن مدى كل مجموعات تعريف الحقول يشمل السجل المناظر له فقط . لاحظ أيضا أن أسماء الحقول مميزة داخل كل سجل طبقا لما هو مطلوب .

## ٢ – تشغيل السجل : 2. PROCESSING A RECORD

والآن ... وبعد أن رأينا كيف تعرف السجلات ، دعنا نعيد انتباهنا لاستخدام السجلات ، أو استخدام مكونات السجلات فى مواقف برمجه تقليدية .

ويشمل أبسط أنواع عمليات تشغيل السجلات تحديد أحد محتويات السجل الآخر . ويتطلب ذلك أن يكون السجلان لهما نفس التكوين تماما .

مثال (١٠ – ١٠)

فيما يلى تخطيطا هيكليا لبرنامج بسكال ، يحتوى على تحديد سجل لآخر ( لاحظ أن هذا التخطيط يستخدم تعريفات السجلات التى سبق تقديمها فى المثال رقم (١٠- ٨) ) .

```
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..80] OF char;
     date = RECORD
              month : 1..12;
              day : 1..31;
              year : 1900..2100
            END;
```

```
        account = RECORD
                      custname : line;
                      custaddress : line;
                      custno : 1..9999;
                      custtype : status;
                      custbalance : real;
                      lastpayment : date
                 END;
VAR preferred,regular : account;

BEGIN
   .
   .
   preferred := regular;
   .
   .
END.
```

ويفترض بالطبع أن عناصر regular تم إدخالها داخل الكمبيوتر ، أو أنها سبق تعريفها قبل عبارة التحديد .

ومن الشائع أكثر تشغيل عناصر السجل الفردية ، بدلا من تشغيل محتويات السجل مع بعضها . ولعمل ذلك يجب أن نكون قادرين على الاتصال بعناصر السجل الفردية . ويتحقق ذلك عن طريق عمل لقب ( أو محدد ) للحقل field designator ، وهو عبارة عن خليط من اسم متغير من نوع السجل ، واسم حقل ، أى أن :

*variable name.field name*

( لاحظ وجود النقطة بين الاسمين ) .

## مثال (١٠-١١)

اعتبر مرة أخرى المتغيرين من نوع السجل regular , preferred المعرفين فى مثال (١٠ – ١٠) . يشار إلى عناصر البيانات الفردية الموجودة داخل كل من هذه السجلات بمحدداته للحقل مثل :

regular . custno , preferred. custname , regular. lastpayment

فمثلا preferred. custname, preferred. custaddress يمثلان متغيرات سلسلة ، كل منها به ٨٠ عنصرا ، ويمثل preferred. custno كمية عددية صحيحة تقع قيمتها بين 9999,1 ويمثل preferred.custtype إحدى الكميات البسيطة متعددة النوع من delinquent , overdue , current . كما يمثل preferred. custbalance كمية عددية حقيقية يعرفها تاريخ السجل ( المزيد من هذا موجود فى المثال التالى ) . ويستخدم أوصاف حقول شبيهه لعناصر regular .

إذا مثل أحد الحقول عنصر بيانات مرتب ، فإن العناصر الفردية لهذا الحقل يمكن الاتصال بها بوضع عنصر البيانات المرتب فى محدد للحقل . وعلى هذا فإذا مامثل الحقل منظومة ، فيمكن الاتصال بأحد عناصر المنظومة بكتابة مايلى :

*variable name.field name [index values]*

وبالمثل إذا مامثل أحد الحقول سجلا محصورا ، فيمكن الاتصال بعنصر فردى من عناصر السجل المحصور كما يلى :

*variable name.field name.sub-field name*

حيث يشير sub - field إلى حقل داخل سجل محصور .

## مثال (١٠-١٢)

اعتبر مرة أخرى المتغيرين من نوع السجل regular, preferred المعرفين فى مثال (١٠ – ١٠) . يشير محدد الحقل preferred. custname [12] إلى الحرف الثانى عشر فى أول حقل من preferred ( أى الحرف الثانى عشر فى المنظومة المضغوطة custname ) .

وبالمثل يشير محدد الحقل regular. lastpayment. month إلى أول كمية عددية صحيحة فى آخر حقل من regular ( أى الكمية الموجودة فى أول حقل فى السجل المحصور المسمى lastpayment ) . يجب أن تكون هذه الكمية عددية صحيحة ، تقع قيمتها داخل المدى الجزئى 1 .. حتى 12 كما يعرفها data ، وهو نوع السجل .

ويمكن استخدام عناصر السجل الفردية بنفس الطريقة مثل المتغيرات المعتادة . والمعالم الخاصة ( وكذلك القيود ) المطبقة على كل عنصر يتم تحديدها بنوع بيانات هذا العنصر . وعلى هذا ... إذا مامثل عنصر سجل كمية من النوع البسيط ، فيمكن أن يظهر على ذلك فى عبارة تحديد أو فى تعبير أو فى مكون تحكم أو عبارة مدخلات أو مخرجات أو كمؤشر داخل إشارة إلى إجراء أو إلى دالة ، وما إلى ذلك طبقا للقيود المطبقة على نوع البيانات الخاص هذا .

وبالمثل إذا مامثل عنصر سجل نوع بيانات مرتب ( مثل المنظومة ، أو أى سجل آخر ) فيمكن استخدامه بنفس الطريقة ، مثل أى عنصر بيانات مرتب من نفس النوع . والقيد الخاص الوحيد هو أن أحد عناصر السجل المضغوط لايمكن أن يمرر إلى إجراء أو إلى دالة إذا ما كان المؤشر الرسمى المناظر هو مؤشراً متغيراً .

## مثال (١٠-١٣)

فيما يلى عدة عبارات غير مرتبطة ببعضها ، تستخدم عناصر سجل فردية . وكل عناصر السجل تؤيد تعريف السجل وتوضيحات متغيرات من نوع السجل المعطاه فى مثال (١٠ – ١٠) ، وذلك مع افتراض التوافقية :

```
regular.custbalance := 0;

regular.custbalance := regular.custbalance - payment;

preferred.custname := regular.custname;

writeln(regular.custname, regular.custaddress);

average := (preferred.custbalance + regular.custbalance)/2;

newbalance := round(preferred.custbalance);

IF preferred.custname [1] = '*'
            THEN preferred.custtype := current;

IF regular.lastpayment.month < 6
      THEN writeln(regular.custno, regular.custbalance)
      ELSE preferred := regular;
```

تتطلب بعض التطبيقات أن يخزن تسلسل السجلات ويشغل بترتيب خاص . وفى مثل هذه الحالات يكون من المقنع تعريف منظومة ذات بعد واحد وعناصرها كلها سجلات . يمكن الاتصال على ذلك بسجل معين كما يلى :

*array name [index value]*

والاتصال بعنصر سجل معين كما يلى :

*array name [index value].field name*

## مـــثال (١٠-١٤)

فيما يلى مثالاً لمنظومة ( وهى list ) تحتوى على ١٠٠ سجلا :

```
TYPE account = RECORD
                    custno : integer;
                    custtype : char;
                    custbalance : real
                 END;
VAR customer : ARRAY [1..100] OF account;
```

وعلى هذا يشير customer [23] إلى السجل رقم (٢٣) ، كما أن customer [23]. custbalance يشير إلى ثالث عنصر بيانات فى السجل رقم (٢٣) ( وهو الموازنة الحالية للعميل رقم ٢٣ ) .

## مثال (١٠-١٥)

فيما يلى مثالاً معقداً أكثر لعدة منظومات ( كل منظومة منها ) ذات بعد واحد ، وعناصرها سجلات .

```
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..80] OF char;
     date = RECORD
               month : 1..12;
               day : 1..31;
               year : 1900..2100
            END;
     account = RECORD
                  custname : line;
                  custaddress : line;
                  custno : 1..9999;
                  custtype : status;
                  custbalance : real;
                  lastpayment : date
               END;
VAR preferred,regular : ARRAY [1..100] OF account;
```

فى هذا المثال يشير preferred [5] إلى خامس سجل فى المنظومة الخامسة ، ويشير regular [82]. custname إلى اسم العميل فى السجل رقم ٨٢ من المنظومة الثانية . وبالمثل يشير regular [14]. lastpayment. year إلى سنة آخر مبلغ دفعه العميل ( أى إلى العنصر الثالث فى السجل المحصور ) فى السجل رقم (١٤) من المنظومة الثانية .

مثـال (١٠-١٦)

فيما يلي عدة عبارات غير مرتبطة ببعضها ، تستخدم عناصر من سجلات من المثال السابق .

```
regular [82].custbalance := 0;

preferred [i].custbalance :=
                    preferred [i].custbalance - payment;

writeln(regular [15].custname, regular [15].custaddress);

IF preferred [count].custname [1] ='*'
            THEN preferred [count].custtype := current;

IF regular [i].lastpayment.month < 6
   THEN writeln(regular [i].custno, regular [i].custbalance)
   ELSE preferred [i] := regular [i];
```

فى هذه الأمثلة المتغيرات فى count , i مفترض أنها متغيرات من النوع الصحيح ، محدد لها قيم مناسبة فى جزء سابق فى البرنامج .

## ٣ – مكون WITH : 3. THE WITH STRUCTURE

تتطلب برامج عديدة معاملة عناصر مختلفة من نفس السجل فى أماكن مختلفة داخل البرنامج . ويمكن أن تصبح الحاجة إلى تحديد محددات مختلفة لحقول مملة ، وعلى أية حال يمكن أن تقلل من سهولة قراءة البرنامج ككل . وفى مثل هذه الحالات يسمح مكون WITH بحذف أسم السجل من محددات الحقول .

والصيغة العامة لمكون WITH هى :

```
WITH record name DO statement
```

أو ، إذا ماوجد سجلان أو أكثر داخل المكون ، فإنها تصبح :

```
WITH record 1 name, record 2 name, . . . , record n name DO statement
```

جزء statement من المكون يشير إلى أى عبارة إجرائية ، والتى يمكن أن تكون هى بنفسها مكونا آخر . داخل هذه العبارة أى إشارة إلى عنصر داخل أحد السجلات المحددة لاتحتاج أن تحتوى على اسم السجل .

مـثال (١٠-١٧)

اعتبر التوضيح التالى :

```
TYPE date = RECORD
               month : 1..12;
               day : 1..31;
               year : 1900..2100
            END;
VAR birthday : date;
```

احدى الطرق لتعديل يوم الميلاد تُكتب :

```
BEGIN
   birthday.month ;= 5;
   birthday.day := 13;
   birthday.year := 1966
END;
```

من الأسهل على أية حال كتابة مايلى :

```
WITH birthday DO BEGIN month := 5; day := 13; year := 1966 END;
```

وكل من العبارتين يحقق نفس الشئ .

## مثـال (١٠-١٨)

فيما يلى مثالا آخر يوضح استخدام مكون WITH . يستخدم هذا المثال توضيحات سبق تقديمها فى مثـال (١٠- ٧ ) . وهذه التوضيحات معادة فى التخطيط الهيكلى التالى للبرنامج .

```
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..80] OF char;
     date = RECORD
               month : 1..12;
               day : 1..31;
               year : 1900..2100
            END;
     account = RECORD
                  custname : line;
                  custaddress : line;
                  custno : 1..9999;
                  custtype : status;
                  custbalance : real;
                  lastpayment : date
               END;
VAR preferred,regular : account;

BEGIN
   .
   .
   WITH regular DO
        BEGIN
           .
           .
           custno := 1262;
           lastpayment.day := 8;
           .
           .
           IF custbalance = 0 THEN custtype := current
                              ELSE custtype := overdue;
           .
           .
        END;
   .
   .
END.
```

لاحظ أن كل إشارات الحقول داخل مكون WITH تشير إلى عناصر من regular . إذا كان ضروريا الإشارة إلى عنصر من عناصر preferred ، فإن محتوى محدد الحقل ( أى custbalance , preferred ) . يجب أن يحدد .

## مثـال (١٠-١٩)

اعتبر التخطيط الهيكلى التالى للبرنامج الذى يستخدم توضيح للمنظومة من نوع السجل الموضحة فى مـثال (١٠ - ١٥) .

```
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..80] OF char;
     date = RECORD
               month : 1..12;
               day : 1..31;
               year : 1900..2100
            END;
     account = RECORD
                  custname : line;
                  custaddress : line;
                  custno : 1..9999;
                  custtype : status;
                  custbalance : real;
                  lastpayment : date
               END;
VAR i : 1..100;
    preferred,regular : ARRAY [1..100] OF account;

BEGIN
   .
   .
   FOR i := 1 TO 100 DO
      WITH regular [i] DO
           BEGIN
              .
              .
              custno := i + 1200;
              lastpayment.day := 8;
              .
              .
              IF custbalance = 0 THEN custtype := current
                                 ELSE custtype := overdue;
              .
              .
           END;
   .
   .
END.
```

قارن هذا المثال مع التخطيط الموضح فى مثال (١٠ - ١٨) .

اذا ماإحتوى مكون WITH على اسمى متغيرين أو أكثر من نوع السجل لها نفس أسماء الحقول ، فمن الممكن

ظهور إبهام فى أى السجلات التى يشار إليها عند الاتصال بحقل داخل المكون . ويحل هذا الإبهام كما يلى :

مكون WITH متعدد السجلات :

```
WITH record 1 name, record 2 name, . . . , record n name DO statement
```

يناظر المكون المتداخل التالى :

```
WITH record 1 name DO
   WITH record 2 name DO
      .
            WITH record n name DO statement
```

إذا ماكان اسم الحقل معتادا لسجلين أو أكثر من السجلات المحددة ، فإن مداه يتطابق مع السجل الداخلى جدا من التداخلات ، وأى إشارة إلى حقل داخل نفس الاسم ، لكنه يصاحب سجلا أخرا يجب أن تحتوى على اسم السجل كجزء من محدد الحقل .

## مثـال (١٠-٢٠)

فيما يلى تخطيطا هيكليا لبرنامج :

```
VAR first : RECORD a,b,c : integer END;
   second : RECORD c,d,e : integer END;
BEGIN
   .
   .
   WITH first, second DO
      BEGIN
         a := 1;
         b := 2;
         first.c := 3;
         c := 4;
         d := 5;
         e := 6
      END;
   .
   .
END.
```

لاحظ أن c هى اسم حقل معتاد مع كل من السجلين . وعلى هذا ... فالإشارة إلى c داخل مكون WITH تشير تلقائيا إلى c الموجودة فى second ، حيث إن second هو اسم السجل الداخلى جدا . وبالإشارة إلى c الموجودة فى first داخل هذا المكون ، يجب أن نحدد محتوى محدد الحقل .

يتسبب المكون السابق فى تحديد القيم التالية للحقول المسماة c .

```
first.c := 3          second.c := 4
```

ويجب استخدام المكون WITH فى الحالات التى يستخدم فيها بحرية . ويسهم استخدامه فى وضوح البرنامج ككل ، وفى بعض الحالات يمكن أن يحسن من كفاءة حسابات البرنامج .

## مثـال (١٠-٢١)

*نظام فواتير العملاء* . اعتبر نظام فواتير بسيطة للعملاء ، حيث يتم إدخال سجلات العملاء فيه داخل الكمبيوتر ، ويتم تجديد موازنة كل عميل لتعكس مدفوعاته الحالية ، وحساب الموازنة الجديدة . ونحسب بالإضافة إلى ذلك الحالة الحالية لكل عميل ونعرضها طبقا لموازنة العميل السابقة ولقيمة الموازنة الحالية .

افرض أن سجل كل عميل يحتوى على إحدى المعلومات التالية : الاسم – وعنوان الشارع – واسم المدينة – واسم الولاية – ورقم الحساب – وحالة الحساب ( حساب جارٍ أو متأخر أو مقصر ) . والموازنة السابقة والمدفوعات الحالية والموازنة الجديدة وتاريخ الدفع .

وفيما يلى توضيح السجل :

```
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..30] OF char;
     date = RECORD
              month : 1..12;
              day : 1..31;
              year : 1900..2100
            END;
     account = RECORD
                 name : line;
                 street : line;
                 city : line;
                 custno : 1..9999;
                 custtype : status;
                 oldbalance : real;
                 newbalance : real;
                 payment : real;
                 paydate : date
               END;
```

تحدد حالة كل حساب بالطريقة التالية : يعتبر الحساب جاريا ، إلا إذا كان :

١ – المبلغ المدفوع حاليا أكبر من صفر ، لكنه أقل ١٠ ٪ من الموازنة القائمة السابقة . يعتبر الحساب فى هذه إلى متأخر overdue .

٢ – يوجد موازنة قائمة . والمبلغ المدفوع حاليا صفر ، وفى هذه الحالة يعتبر الحساب مقصراً delinquent .

وسوف تخزن كل السجلات كعناصر لمنظومة ذات بعد واحد .

وعلى هذا ... تصبح خطوات البرنامج الشامل كما يلى :

١ – حدد عدد الحسابات ( أى عدد السجلات ) التى سيتم تشغيلها .

٢ – اقرأ العناصر التالية لكل سجل :

أ – الاسم .

ب – الشارع .

جـ - المدينة .

د - رقم الحساب .

هـ - الموازنة السابقة .

و - المبلغ المدفوع حاليا .

ز - تاريخ الدفع .

٣ - بعد قراءة كل السجلات داخل الكمبيوتر ، شغل كل سجل بالطريقة التالية :

أ - قارن المبلغ المدفوع حاليا مع الموازنة السابقة ، وحدد حالة الحساب المناسبة .

ب - احسب موازنة الحساب الجديدة بطرح المبلغ المدفوع حاليا من الموازنة السابقة ( وتحدد الموازنة السابقة الرصيد الدائن ) .

٤ - بعد تشغيل كل السجلات ، اكتب المعلومات التالية لكل سجل :

أ - الاسم .

ب - رقم الحساب .

جـ - الشارع .

د - المدينة .

هـ - الموازنة القديمة .

و - القيمة المدفوعة حاليا .

ز - الموازنة الجديدة .

حـ - حالة الحساب .

دعنا نكتب البرنامج على هيئة أجزاء مع ثلاثة اجراءات منفصلة لقراءة بيانات المدخلات ، وتشغيل البيانات ، وكتابة المخرجات على التوالى . وكل إجراء من هذه الإجراءات مباشر ، ولايتطلب تفصيلا أكثر . سوف تدخل المجموعة الرئيسية السجلات ، ثم تتصل بالإجراءات المناسبة .

وفيما يلى محتويات البرنامج :

```
PROGRAM billing1(input,output);

(* THIS PROGRAM ILLUSTRATES THE USE OF RECORDS
            IN A SIMPLE CUSTOMER BILLING SYSTEM *)

TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..30] OF char;
     date = RECORD
               month : 1..12;
               day : 1..31;
               year : 1900..2100
            END;
```

(تكملة البرنامج فى الصفحة التالية)

```
     account = RECORD
                 name : line;
                 street : line;
                 city : line;
                 custno : 1..9999;
                 custtype : status;
                 oldbalance : real;
                 newbalance : real;
                 payment : real;
                 paydate : date
              END;
VAR customer : ARRAY [1..100] OF account;
    i,n : 1..100;

PROCEDURE readinput;
(* Read input data for each record *)
VAR count : 1..30;
    slash : char;
BEGIN
   FOR i := 1 TO n DO
      WITH customer [i] DO
         BEGIN
            writeln;
            writeln('Customer no. ',i:3);
            write('  Name:   ');
            count := 1;
            REPEAT
               read(name [count]);
               count := count + 1
            UNTIL eoln;
            readln;
            write('  Street: ');
            count := 1;
            REPEAT
               read(street [count]);
               count := count + 1
            UNTIL eoln;
            readln;
            write('  City:   ');
            count := 1;
            REPEAT
               read(city [count]);
               count := count + 1
            UNTIL eoln;
            readln;
            write('  Account number: ');
            readln(custno);
            write('  Previous balance: ');
            readln(oldbalance);
            write('  Current payment: ');
            readln(payment);
            write('  Payment date (mm/dd/yyyy): ');
            WITH paydate DO
               read(month,slash,day,slash,year);
            readln
         END
END;   (* readinput *)
```

(تكملة البرنامج فى الصفحة التالية)

```
PROCEDURE processdata;
(* Determine status and calculate a new balance for each record *)
BEGIN
   FOR i := 1 TO n DO
      WITH customer [i] DO
         BEGIN
            custtype := current;
            IF (payment > 0) AND (payment < 0.1*oldbalance)
                THEN custtype := overdue;
            IF (oldbalance > 0) AND (payment = 0)
                THEN custtype := delinquent;
            newbalance := oldbalance - payment
         END
END;  (* processdata *)

PROCEDURE writeoutput;
(* Write out current information for each record *)
BEGIN
   FOR i := 1 TO n DO
      WITH customer [i] DO
         BEGIN
            writeln;
            write('Name:    ',name);
            writeln('   Account number: ',custno:4);
            writeln('Street: ',street);
            writeln('City:    ',city);
            writeln;
            write('Old balance: ',oldbalance:7:2);
            write('   Current payment: ',payment:7:2);
            writeln('   New balance: ',newbalance:7:2);
            writeln;
            write('Account status: ');
            CASE custtype OF
               current    : writeln('CURRENT');
               overdue    : writeln('OVERDUE');
               delinquent : writeln('DELINQUENT')
            END;
            writeln
         END
END;  (* writeoutput *)

BEGIN  (* main action statements *)
   writeln('CUSTOMER BILLING SYSTEM');
   writeln;
   write('How many customers are there? ');
   readln(n);
   readinput;
   processdata;
   writeoutput
END.
```

لاحظ أن مكون WITH موجود فى كل إجراء من الإجراءات . وفى الواقع يحتوى readinput على مكون WITH مـزدوج ( أى متداخل ) يعد العناصر الفردية للسجل المحصور paydate ، والموجود داخل كل سجل من سجلات العملاء .

افرض الآن أن البرنامج يستخدم فى تشغيل أربعة سجلات لعملاء وهميين . وفيما يلى حوار المدخلات ، مع وضع خط تحت استجابة المستفيد ..

```
CUSTOMER BILLING SYSTEM

How many customers are there? 4

Customer no.   1
   Name:  Richard L. Warren
   Street: 123 Vistaview Drive
   City:  Denver, CO
   Account number: 4208
   Previus balance: 247.88
   Current payment: 25.00
   Payment date (mm/dd/yyyy): 6/14/1983

Customer no.   2
   Name:  Marcia Korenstein
   Street: 4383 Affluent Avenue
   City:  Beechview, OH
   Account number: 2219
   Previous balance: 135.00
   Current payment: 135.00
   Payment date (mm/dd/yyyy): 8/10/1983

Customer no.   3
   Name:  Mark Singer
   Street: 1787 Larynx Lane
   City:  Indianapolis, IN
   Account number: 8452
   Previous balance: 387.42
   Current payment: 35.00
   Payment date (mm/dd/yyyy): 7/4/1983

Customer no.   4
   Name:  Phyllis W. Smith
   Street: 1000 Great White Way
   City:  New York, NY
   Account number: 711
   Previous balance: 260.00
   Current payment: 0
   Payment date (mm/dd/yyyy): 11/27/1983
```

عند ذلك ينتج البرنامج بيانات المخرجات التالية كاستجابة للمدخلات سالفة الذكر .

```
Name:   Richard L. Warren   Customer number: 4208
Street: 123 Vistaview Drive
City:   Denver, CO
```

(التكملة فى الصفحة التالية)

```
Old balance:  247.88   Current payment:   25.00   New balance:  222.88

Account status: CURRENT


Name:   Marcia Korenstein   Customer number: 2219
Street: 4383 Affluent Avenue
City:   Beechview, OH

Old balance:  135.00   Current payment:  135.00   New balance:    0.00

Account status: CURRENT

Name:   Mark Singer   Customer number: 8452
Street: 1787 Larynx Lane
City:   Indianapolis, IN

Old balance:  387.42   Current payment:   35.00   New balance:  352.42

Account status: OVERDUE


Name:   Phyllis W. Smith   Customer number:  711
Street: 1000 Great White Way
City:   New York, NY

Old balance:  260.00   Current payment:    0.00   New balance:  260.00

Account status: DELINQUENT
```

من وجهة النظر العملية ، فإن طبيعة هذا البرنامج غير واقعية بعض الشئ . وبصفة خاصة ... فهو يعطى شعورا بسيطا بإدخال محتويات سجل العميل وتنفيذ حسابات بسيطة ، ثم كتابة سجل مجدد ، دون حفظ نسخة دائمة للمعلومات الجديدة داخل الكمبيوتر ( أو فى أى تخزين ثانوى آخر ) .

وسوف نرى فى الفصل التالى كيف يمكن حفظ سجلات العملاء حفظا دائما فى ملفات بيانات تلغى الحاجة إلى إعادة إدخال البيانات عندما يكون هناك حاجة إلى تجديدها .

## 4. VARIANT RECORDS — ٤ - سجلات متغيرة

حتى الآن لم نعتبر إلا سجلات تظل مكوناتها ثابتة أثناء تنفيذ البرنامج . من الممكن أيضا - على أية حال - أن نعرف سجل يمكن أن تتغير محتوياته ( أو جزء من محتوياته ) داخل البرنامج ، طبقا للقيمة التى تحدد لبعض الحقول الخاصة الموجودة داخل السجل ( حقل الشعار tag field ) مثل هذا النوع من السجلات يعرف بأنه سجل متغير Variant record ( أو الجزء المتغير Variant part من السجل ) .

والصيغة العامة لتعريف نوع السجل تحتوى على ( كل من ) جزء ثابت ، وجزء متغير كما يلى :

```
TYPE record name = RECORD
                        fixed field 1;
                        fixed field 2;
```

(التكملة فى الصفحة التالية)

```
CASE tag field identifier : type OF
    case label 1 : variant fieldlist 1;
    case label 2 : variant fieldlist 2;
        .
        .
        .
END
```

كل قائمة حقل focld list يعبر عنها فى صورة تعداد الحقل ، أى :

(*variant field 1*, *variant field 2*, . . . , *variant field n*)

يجب أن يصاحب حقل الشعار نوع بيانات ترتيبى سبق تعريفه . ويجب أن تظهر كل قيمة لهذا النوع الترتيبى مرة واحدة فقط كعنوان حالة Case label ، وذلك فى جزء لاحق للمتغير فى تعريف السجل . ويحتوى عل ذلك الجزء النشط من السجل على الحقول التى تناظر بعض عناوين الحالة الخاصة كما هو محدد لمعرف حقل الشعار .

كما يمكن أن يوجد جزء متغير فقط فى أى سجل ، ويجب أن يتبع الجزء الثابت من السجل دائما .

## مثـال (١٠-٢٢)

فيما يلى مثالا لتعريف سجل يحتوى على كل من الجزء الثابت والجزء المتغير .

```
TYPE item = (stereo,tv);
     name = PACKED ARRAY [1..80] OF char;
     inventory = RECORD
                    stockno : 1..20000;
                    supplier : name;
                    quantity : integer;
                    CASE itemtype : item OF
                       stereo : (power : 1..1000);
                           tv : (tubesize : 1..25; color : char)
                 END;
 VAR stockitem,backorder : inventory;
```

ونوع السجل مسمى inventory . يحتوى الجزء الثابت منه على الحقول stockno , supplier , quantity ، ويحتوى الجزء المتغير على حقل power أو الحقول tubesize , color طبقا للقيمة التى تحدد لمعرف حقل شعار هو item type . إذا ماحدد stereo بصفة خاصة لـ item type فعلى هذا يحتوى الجزء المتغير من السجل على الحقل power ، أما إذا ماتحدد item type > tv فيحتوى – على هذا – الجزء المتغير على الحقول tubesize , color .

لاحظ أننا شملنا توضيحا لمتغيرين من نوع السجل ، وهما stockitem , backorder . كل من هذين المتغيرين يمثل سجلا من نفس نوع inventory .

ويمكن الاتصال بالعناصر الفردية للسجل المتغير وبمعرف حقل شعار الطريقة المستخدمة مع العناصر الفردية

للسجل الثابت ، وذلك بكتابة مايلي :

```
variable name.field name

variable name.field name.sub-field name
```

أو كتابة مايلي :

```
variable name.field name [index values]
```

وذلك طبقا لنوع البيانات الخاص بالعنصر الفردي للسجل .

## مثـال (١٠-٢٣)

فيما يلي عدة عبارات غير مرتبطة ببعضها ، تستخدم عناصر فردية من سجل . وتشير كل العبارات إلى عناصر لمتغيرات من نوع السجل المعرفة في مثال (١٠ - ٢٢) .

```
stockitem.stockno := 12345;

backorder.quantity := 15;

stockitem.itemtype := stereo;

stockitem.power := 150;

writeln(backorder.stockno,backorder.power);

IF stockitem.quantity < 24 THEN
   BEGIN
      writeln(stockitem.supplier);
      writeln(stockitem.tubesize,stockitem.color)
   END;
```

لاحظ أن بعض العبارات تستخدم حقولا ثابتة ، بينما تستخدم عبارات أخرى حقولا متغيرة . وتحتوى العبارتان الأخيرتان على إشارات لحقول ثابتة وحقول متغيرة . لاحظ أيضا أنه تحددت قيمة لمعرف حقل شـعار في العبـارة الثالثة .

يبين مثال (١٠ - ٢٢) معالم معينة يجب أن يكون المبرمج حريصا من ناحيتها . أولا : يجب أن تكون أسماء الحقول داخل السجل ، بما فيها أسماء الحقول المتغيرة فردية . ( وفي بعض التطبيقات يوجد إغراء باستخدام نفس اسم الحقل في قائمتين أو أكثر للحقول المتغيرة ) . ثانيا : يجب تحديد قيمة مناسبة لمعرف حقل شعار إذا ماكان مطلوبًا الاتصال بحقل متغير واحد . وعلى هذا ... إذا ما أريد الاتصال في مثال (١٠ - ٢٢) بـ stockitem. power ، فيجب أن يمثل stockitem . itemtype بأنه stereo .

يمكن استخدام مكون WITH مع سجلات من النوع المتغير ، مثل استخدامها تماما مع سـجلات النوع الثابت ( انظر القسم ٣ من هذا الفصل ) . وهذا يميل إلى تقليل احتمالات الخطأ ، وتحسين القراءة الشاملة للبرنامج .

## مثـال (١٠-٢٤)

افرض أننا نريد تجديد حقولا محددة فى سجل stockitem المعرف فى مثال (١٠ - ٢٢) . يمكن تحقيق ذ
بسهولة باستخدام مكون WITH كما يلى :

```
WITH stockitem DO
     BEGIN
        quantity := quantity - sales;
        IF quantity < 0 THEN backorder.quantity := -quantity;
        IF itemtype = stereo THEN writeln(power)
                             ELSE writeln(tubesize,color)
     END;
```

لاحظ أن هذا المثال يحتوى على استخدام حقل ثابت ( quantity ) ، وحقل شعار ( item type ) ، وحـ
عديدة متغيرة ( color , tubesize , power ) . لاحظ أيضا الإشارة لحقل واحد داخل سجل آخر ( backorder
وتكتب الإشارة إلى الحقل فى صورتها الكاملة ( أى على هيئة backorder. quantity ) حيث إن عبارة WITH لاتنـ
على هذا السجل .

وهناك العديد من الاختلافات فى شكل السجل العام يمكن ذكرها . فمن الممكن - على سبيل المثال - تعر
سجل لايحتوى على جزء ثابت . أكثر من هذا ... يمكن أن تكون قائمة الحقول المتغيرة فارغة إذا ماكانت هناك رغبة
ذلك . ويحدد هذا بواسطة فئة فارغة من الأقواس ، يليها عنوان الحالة المناسب . وكل من هاتين الحالتين موضح
المثال التالى :

## مثـال (١٠-٢٥)

اعتبر تعبير السجل المتغير التالى :

```
TYPE status = (single,married,divorced,widowed);
     background = RECORD
                     CASE maritalstatus : status OF
                        single : ();
                        married : (children : 0..10);
                        divorced,widowed : (children : 0..10;
                                            remarried : boolean)
                  END;
VAR employees : ARRAY [1..100] OF background;
```

فى هذا المثال employees هى منظومة ذات بعد واحد ، وعناصرها سجلات من نفس نوع ackground
لاحظ أن تعريف هذا السجل الخاص لايحتوى على جزء ثابت . لاحظ أيضا أن أول حقل متغير list ( المناظر لعنـ
الحالة single ) فارغ .

كما نرى أيضا أن آخر قائمة حقل متغير تناظر عنوانى الحالة المختلفين widowed , divorced . وأ
لاحظ أن الحقل children مشترك مع كل من قائمتى الحقول المتغيرتين الثانية والثالثة ( أى مع عناوين الحالة orced
, widowed, married ) . وتقترح طبيعة هذا المثال أن الجزء الثابت من السجل يمكن أن يكون موجودا أيضا ، محـ
على اسم name وعنوان address ورقم الموظف employee number .. الخ . ويمكن ( للقارئ ) إضافة هذه الحـ
ببساطة . وهناك موقف آخر يحتاج لمناقشة ، وهو وجود معرف حقل شعار داخل تعريف السجل . وتواجد معرف

شعار ليس ضروريا . فإذا لم يتواجد ، فتحدد الحقول المتغيرة النشطة ضمنيا بواسطة حقل متغير يمكن الاتصال به . وبصفة خاصة الإشارة إلى حقل متغير خاص تنشط عنوان حالة خاصة ، وبذلك تصبح كل الحقول المتغيرة المصاحبة لعنوان الحالة هذا نشطة . وتقترح البرمجة الجيدة – على أية حال – عدم استخدام هذه السمة بصفة عامة ، خاصة فى حالة المبرمجين المبتدئين .

## مثـال (١٠-٢٦)

اعتبر التخطيط الهيكلى للبرنامج التالى :

```
TYPE item = (stereo,tv);
     name = PACKED ARRAY [1..80] OF char;
     inventory = RECORD
                    stockno : 1..20000;
                    supplier : name;
                    quantity : integer;
                    CASE item OF
                       stereo : (power : 1..1000);
                           tv : (tubesize : 1..25;
                                   color : char)
                END;
VAR stockitem,backorder : inventory;
BEGIN
   WITH stockitem DO
      BEGIN
         .
         .
         tubesize := 19;
         color := 'Y';
         .
         .
      END
END.
```

لاحظ أن تعريف السجل يحتوى على نوع حقل شعار ( وهو item ) ، وليس على معرف حقل شعور . وعلى هذا ... فالإشارات إلى color , tubesize داخل مجموعة الإجراءات الرئيسية تشمل أن قائمة الحقل المتغير الثانية نشطة ( أى أنها تناظر عنوان الحالة tv ) .

وبالرغم من أن أحد السجلات يمكن أن يكون له جزء ثابت واحد ، وجزء متغير واحد فقط ، فمن الممكن أن يوجد تعريف متغير واحد داخل تعريف آخر . وعلى هذا ... فيمكننا إنتاج حقولا داخل سجل فردى .

## مثـال (١٠-٢٧)

فيما يلى صورة أخرى لتعريف السجل الموجود فى مثال (٢٢ – ١٠) . (أنظر الصفحة التالية)

```
TYPE item = (stereo,tv);
     style = (portable,stationary);
     name = PACKED ARRAY [1..80] OF char;
     inventory = RECORD
                    stockno : 1..20000;
                    supplier : name;
                    quantity : integer;
                    CASE itemtype : item OF
                       stereo : (power : 1..1000);
                           tv : (tubesize : 1..25;
                                 color : char;
                                 CASE size : style OF
                                    portable : (weight : 1..100;
                                                voltage : 1..220);
                                    stationary : ())
                 END;
VAR stockitem,backorder : inventory;
```

لاحظ قائمة الحقول المتغيرة الثانية ( المناظرة لعنوان الحالة tv ) . تحتوى قائمة الحقول هذه على تعريف متغير آخر يعتمد على القيمة المحددة لمعرف حقل شعار ، وهو size . فإذا كان size يمثل portable . فإن قائمة الحقول المتغير tv تحتوى على الحقلين color , tubesize فقط .

يجب أن يلاحظ القارئ أن التعريف المتغير الداخلى جدا له حقل فردى ( size ) يصاحب نوع بيـاناته المتـعدد ( style ) . لاحظ أيضا قائمة الحقل الفارغة المناظرة لعنوان الحالة المتداخل stationary . وأخيرا عليك أن تلاحظ أن الحقول المتغيرة المصاحبة لـ tv تتبع الحقول الثابتة كما هو مطلوب .

ويجب أن يكون مفهوما أن مجموعة جديدة من الحقول المتغيرة سوف تصبح نشطة عندما يتم تحديد قيمة جديدة لمعرف حقل شعار . أكثر من هذا ... إذا لم يتواجد معرف شعار ، فسوف تصبح مجموعة جديدة من الحقول المتغيرة variants نشطة كنتيجة للإشارة إلى حقل متغير غير نشط . وأى تغير غير مطلوب فى الحقول المتغيرة النشطة يجب تجنبه على ذلك . ويجب أن يكون المبرمج حريصا من ناحية هذه المشكلة فى البرامج التى تستخدم إجراءات أو دوال .

ولكى يقل احتمال مثل هذه الأخطاء ، لايسمح بتغيير حقل متغير variant تحت الظروف التالية :

١ – داخل مكون WITH إذا ظهر اسم الحقل المتغير فى عبارة WITH .

٢ – عندما يمر الحقل المتغير إلى إجراء أو دالة ، كمؤشر متغير فعلى .

أكثر من ذلك ... لايمكن أن يمر معرف حقل شعار إلى إجراء أو دالة كمؤشر متغير فعلى .

## مثـال (١٠–٢٨)

مراقبة المخزون . دعنا نطور الآن نظام مراقبة مخزون بسيطًا يشبه نظام فواتير العملاء الموضح فى مثال (١٠ – ٢١) . وسوف نستخدم الآن سجلات تحتوى على كل من الجزء الثابت والجزء المتغير . وبصفة خاصة اعتبر نظامًا يتتبع نوعية من السلع ، وهى أجهزة ستريو stereos ، وأجهزة التليفزيون television . يتم ادخال المعلومات التالية دوريا داخل الكمبيوتر : رقم التخزين stock number ( وهو رقم غير سالب ) ، ونوع العنصر item type ( "s" إذا كان جهاز ستريو ، و "t" لجهاز التليفزيون ) واسم المورد supplier's name ، ومسئول المخزون الأصلى ( رقم غير سالب ) ، والتغيير الحالى فى مسئول المخزون ( رقم سالب أو موجب أو صفر ) .

ذلك بالإضافة إلى بعض المعلومات الوصفية التى تضاف إلى كل عنصر ، وتكون طبيعتها معتمدة على نوع العنصر نفسه .

فبالنسبة لجهاز ستريو ، يتم إدخال مستوى الطاقة ( رقم صحيح يتراوح من ١ إلى ١٠٠٠ ) . وبالنسبة لجهاز التليفزيون ، يكون هناك تحديد إذا ماكان الجهاز قابلاً للنقل "y" أم لا "n" وكذلك حجم الشاشة ( رقم صحيح يتراوح من ١ إلى ٢٥ ) وتحديد ماإذا كان الجهاز ملوناً "y" أو لا "n" ، وكذلك إذا كان الجهاز قابلا للنقل ، فيحدد الوزن ( من ١ إلى ١٠٠ ) والفولت ( من ١ إلى ٢٢٠ ) يتم إدخالها أيضا .

وتوضيح السجل المناسب موضح كما يلى :

```
TYPE item = (stereo,tv);
     style = (portable,stationary);
     name = PACKED ARRAY [1..80] OF char;
     inventory = RECORD
                    stockno : 0..20000;
                    supplier : name;
                    quantity : integer;
                    CASE itemtype : item OF
                       stereo : (power : 1..1000);
                           tv : (tubesize : 1..25;
                                 color : char;
                                 CASE size : style OF
                                    portable : (weight : 1..100;
                                                voltage : 1..220);
                                    stationary : ())
                 END;
VAR stockitem,backorder : ARRAY [1..100] OF inventory;
```

لاحظ أن سجل inventory يحتوى على أجزاء متغيرة متداخلة . لاحظ أيضا أننا عرفنا منظومتين مختلفتين ، هما backorder , stockitem تحتويان على سجلات من نفس نوع inventory . يحتوى stockitem على سجلات مخزون كما تم إدخالها فى الكمبيوتر ، ويحتوى backorder على معلومات عن العناصر التى يحدث لها أوامر خلفية فقط ( أى العناصر التى بها عجز ) .

وتحتوى العملية الكاملة على الخطوتين التاليتين :

١ - أقرأ كل بيانات المدخلات لكل سجل ، واضبط مستوى المخزون طبقا لذلك .

٢ - بعد قراءة كل السجلات داخل الكمبيوتر ، وتضبيط مستويات المخزون طبقا لها ، اكتب السجلات موضحا مستويات المخزون الجديدة .

وبدلا من تحديد عدد السجلات مقدما - كما فعلنا مع نظام الفواتير الموضح فى مثال (٢١ - ١٠) - دعنا نستمر فى إدخال سجلات جديدة حتى يتم إدخال قيمة صفر فى رقم التخزين . وعلى هذا ... تبدو المجموعة الإجرائية الأساسية على النحو التالى :

```
BEGIN
   writeln('I N V E N T O R Y   C O N T R O L   S Y S T E M');
   writeln;
   writeln('              D A T A   I N P U T');
```

(أنظر الصفحة التالية)

```
    writeln;
    writeln('  When finished, enter 0 for the stock number');
    writeln;
    i := 0;
    REPEAT
       i := succ(i);
       readinput;
    UNTIL stockitem [i].stockno = 0;
    n := pred(i);
    IF n > 0 THEN writeoutput
 END.
```

لاحظ أن i هى عداد سجلات من النوع العددى الصحيح تتراوح قيمته من o إلى n ، حيث n هى إجمالى عدد السجلات التى يتم إدخالها . ( وتحدد آخر قيمة i بأنها n ) ، كما أن writeoutput , readinput إجراءان ينتج عنهما قراءة بيانات المدخلات داخل الكمبيوتر ، وتضبيط مستوى المخزون ، وكتابة البيانات الجديدة خارج الكمبيوتر على التوالى .

والإجراء read input متداخل ، وبه عدد من الملقنات ( التقليدية ) المناسبة ، والبرمجة مباشرة ، بالرغم من أنها طويلة بعض الشئ . وعلى هذا يكون لدينا مايلى :

```
PROCEDURE readinput;
(* Enter input data for one record *)
VAR count : 1..80;
BEGIN
   WITH stockitem [i] DO
      BEGIN
         writeln;
         write('Stock no.: ');
         readln(stockno);
         IF stockno > 0 THEN
            BEGIN
               write('Item type (s/t): ');
               readln(designator);
               IF (designator = 's') OR (designator = 'S')
                  THEN BEGIN (* stereo *)
                          itemtype := stereo;
                          write('Power level: ');
                          readln(power)
                       END

            ELSE BEGIN (* tv *)
                    itemtype := tv;
                    write('Portable TV? (y/n) ');
                    readln(designator);
                    IF (designator = 'y') OR (designator = 'Y')
                       THEN BEGIN
                               size := portable;
                               write('Weight: ');
                               readln(weight);
                               write('Voltage: ');
                               readln(voltage)
                            END
```

(التكملة فى الصفحة التالية)

```
                    ELSE size := stationary;
                 write('Tube size: ');
                 readln(tubesize);
                 write('Color: (y/n) ');
                 readln(color)
              END;
        write('Supplier: ');
        count := 1;
        REPEAT
           read(supplier [count]);
           count := succ(count)
        UNTIL eoln;
        write('Original inventory level:  ');
        readln(quantity);
        write('Change in inventory level: ');
        readln(change);
        quantity := quantity + change;
        IF quantity < 0 THEN
           BEGIN
              backorder [i] := stockitem [i];
              backorder [i].quantity := abs(quantity);
              quantity := 0
           END
      END   (* IF stockno > 0 *)
   END   (* WITH stockitem [i] *)
END;   (* readinput *)
```

المحدد فى هذا الاجراء متغير شامل من النوع الحرفى ، يعرف نوع المدخلات . كما أن change متغير شامل من النوع الصحيح ، يمثل التغير فى مستوى المخزون .لاحظ أن change يمكن أن يأخذ قيما موجبة أو سالبة طبقا لما إذا كان هناك سحب من المخزون ، أو إعادة ملء المخزون . لاحظ أيضا أن هذا الإجراء مكتوب للحصول على سجل واحد فى نفس الوقت ( حيث إنه غير معروف مسبقا عدد السجلات التى سبق إدخالها فى الكمبيوتر ) .

وهناك سمة ملحوظة أخرى خاصة بإجراء readinput ، وهى حقيقة أن تجديد المخزون يحدث بمجرد إدخال البيانات الجديدة . فإذا كان مستوى المخزون الجديد سالبا ، فتنتقل المعلومات من stockitem [i] إلى backorder [i] ويخزن العجز كقيمة موجبة . واستخدام backorder [i] ليس ضروريا ، ويضيف القليل فى واقع الأمر إلى البرنامج ، فيما عدا إنه يقترح كيف يمكن حفظ مثل هذا الازدواج من السجلات المتناظرة فى برنامج أكثر واقعية ( المزيد عن ذلك يذكر فيما بعد ) .

دعنا نعتبر الآن الإجراء writeoutput . هذا الإجراء ليس فى حاجة لأن يكون متداخلا ، ويمكنه تشغيل كل السجلات فى وقت واحد . وعلى هذا يمكننا كتابته كما يلى :

```
PROCEDURE writeoutput;
(* Write output data for all records *)
BEGIN
   writeln('I N V E N T O R Y   C O N T R O L   S Y S T E M');
   writeln;
   writeln('      C U R R E N T   I N V E N T O R Y');
   writeln;
   FOR i := 1 TO n DO
      WITH stockitem [i] DO
```

(التكملة فى الصفحة التالية)

```
    BEGIN
      writeln;
      writeln('Stock no.: ',stockno:5);
      CASE itemtype OF
         stereo : writeln('Stereo: ',power:4,' watts');
             tv : BEGIN
                     IF size = portable
                        THEN write('Portable ');
                     IF (color = 'y') OR (color = 'Y')
                        THEN write('Color TV')
                        ELSE write('B & W TV');
                     writeln('   ',tubesize:2,'-inch tube');
                     IF size = portable THEN
                        BEGIN
                           write('Weight: ',weight:3,' pounds    ');
                           writeln('Voltage: ',voltage:3,' volts')
                        END
                  END   (* tv *)
      END;   (* CASE itemtype *)
      writeln('Supplier: ',supplier);
      IF quantity > 0
         THEN writeln('Present inventory: ',quantity:3)
         ELSE writeln('Quantity backordered: ', backorder [i].quantity:3);
      writeln
    END   (* WITH stockitem [i] *)
END;   (* writeoutput *)
```

لاحظ أن أحد حقول backorder [i] يكتب فى حالة ماإذا كان هناك عجز فى المخزون . وفيما يلى محتويات برنامج البسكال كاملة .

```
PROGRAM inventory1(input,output);
(* THIS PROGRAM ILLUSTRATES THE USE OF VARIANT RECORDS
                IN A SIMPLE INVENTORY CONTROL SYSTEM *)
TYPE item = (stereo,tv);
     style = (portable,stationary);
     name = PACKED ARRAY [1..80] OF char;
     inventory = RECORD
                    stockno : 0..20000;
                    supplier : name;
                    quantity : integer;
                    CASE itemtype : item OF
                       stereo : (power : 1..1000);
                           tv : (tubesize : 1..25;
                                 color : char;
                                 CASE size : style OF
                                    portable : (weight : 1..100;
                                                voltage : 1..220);
                                    stationary : ())
                 END;
```

(تكملة البرنامج فى الصفحة التالية)

```
VAR stockitem,backorder : ARRAY [1..100] OF inventory;
    designator : char;
    i,n,change : integer;

PROCEDURE readinput;
(* Read input data for one record *)
VAR count : 1..80;
BEGIN
   WITH stockitem [i] DO
      BEGIN
         writeln;
         write('Stock no.: ');
         readln(stockno);
         IF stockno > 0 THEN
            BEGIN
               write('Item type (s/t): ');
               readln(designator);
               IF (designator = 's') OR (designator = 'S')
                  THEN BEGIN (* stereo *)
                          itemtype := stereo;
                          write('Power level: ');
                          readln(power)
                       END
                  ELSE BEGIN (* tv *)
                          itemtype := tv;
                          write('Portable TV? (y/n) ');
                          readln(designator);
                          IF (designator = 'y') OR (designator = 'Y')
                             THEN BEGIN
                                     size := portable;
                                     write('Weight: ');
                                     readln(weight);
                                     write('Voltage: ');
                                     readln(voltage)
                                  END
                             ELSE size := stationary;
                          write('Tube size: ');
                          readln(tubesize);
                          write('Color: (y/n) ');
                          readln(color)
                       END;
               write('Supplier: ');
               count := 1;
               REPEAT
                  read(supplier [count]);
                  count := succ(count)
               UNTIL eoln;
               write('Original inventory level:  ');
               readln(quantity);
               write('Change in inventory level: ');
               readln(change);
               quantity := quantity + change;
               IF quantity < 0 THEN
                  BEGIN
                     backorder [i] := stockitem [i];
                     backorder [i].quantity := abs(quantity);
                     quantity := 0
                  END
```

(تكملة البرنامج فى الصفحة التالية)

```
            END   (* IF stockno > 0 *)
        END   (* WITH stockitem [i] *)
    END;  (* readinput *)

  PROCEDURE writeoutput;
  (* Write output data for all records *)
  BEGIN
     writeln('I N V E N T O R Y   C O N T R O L   S Y S T E M');
     writeln;
     writeln('      C U R R E N T   I N V E N T O R Y');
     writeln;
     FOR i := 1 TO n DO
        WITH stockitem [i] DO
           BEGIN
              writeln;
              writeln('Stock no.: ',stockno:5);
              CASE itemtype OF
                 stereo : writeln('Stereo: ',power:4,' watts');
                     tv : BEGIN
                             IF size = portable
                                THEN write('Portable ');
                             IF (color = 'y') OR (color = 'Y')
                                THEN write('Color TV')
                                ELSE write('B & W TV');
                             writeln('   ',tubesize:2,'-inch tube');
                             IF size = portable THEN
                                BEGIN
                                   write('Weight: ',weight:3,' pounds    ');
                                   writeln('Voltage: ',voltage:3,' volts')
                                END
                          END   (* tv *)
              END;   (* CASE itemtype *)
              writeln('Supplier: ',supplier);
              IF quantity > 0
                 THEN writeln('Present inventory: ',quantity:3)
                 ELSE writeln('Quantity backordered: ', backorder [i].quantitv:3);
              writeln
           END   (* WITH stockitem [i] *)
  END;   (* writeoutput *)

  BEGIN   (* main action statements *)
     writeln('I N V E N T O R Y   C O N T R O L   S Y S T E M');
     writeln;
     writeln('              D A T A   I N P U T');
     writeln;
     writeln('  When finished, enter 0 for the stock number');
     writeln;
     i := 0;
     (* read input records *)
     REPEAT
        i := succ(i);
        readinput;
     UNTIL stockitem [i].stockno = 0;
     n := pred(i);
     (* write output records *)
     IF n > 0 THEN writeoutput
  END.
```

افرض أن البرنامج استخدم الآن لتشغيل ثلاثة سجلات توضيحية . يبدو حوار المدخلات على النحو التالى :

```
I N V E N T O R Y   C O N T R O L   S Y S T E M

D A T A   I N P U T

When finished, enter 0 for the stock number

Stock no.: 12417
Item type (s/t): s
Power level: 150
Supplier: House of Audio
Original inventory level:  200
Change in inventory level: -12

Stock no.: 912
Item type (s/t): t
Portable TV? (y/n) n
Tube size: 23
Colour: (y/n) y
Supplier: Ace Radio & TV
Original inventory level:  8
Change in inventory level: 72

Stock no.: 644
Item type (s/t): t
Portable TV? (y/n) y
Weight: 25
Voltage: 110
Tube size: 12
Color: (y/n) n
Supplier: Crazy Harry's
Original inventory level:  12
Change in inventory level: -15

Stock no.: 0
```

فى هذا الحوار يقدم الكمبيوتر ملقنات ، ويقوم المستفيد بإدخال المعلومات المطلوبة ( استجابات المستفيد موضوع تحتها خط ) .

لاحظ أن حوار المدخلات يستمر حتى تدخل قيمة لرقم المخزون stock number مساوية صفرا .

وينتج البرنامج المخرجات التالية كاستجابة لبيانات المدخلات :

```
INVENTORY  CONTROL  SYSTEM

     CURRENT  INVENTORY

Stock no.: 12417
Stereo:  150 watts
Supplier: House of Audio
Present inventory: 188

Stock no.:   912
Color TV   23-inch tube
Supplier: Ace Radio & TV
Present inventory:  80

Stock no.:   644
Portable B & W TV   12-inch tube
Weight:  25 pounds    Voltage: 110 volts
Supplier: Crazy Harry's
Quantity backordered:
```

وعلى القارئ أن يتذكر مرة أخرى أن هذا البرنامج – مثل برنامج فواتير العملاء المقدم فى مثال (١٠ – ٢١) – مبسط جدا ، حيث إنه لايحفظ سجلات المخزون دائما داخل ملف بيانات . وفى واقع الأمر ... فإن كل ملفات نظم مراقبة المخزون تحتوى على مثل هذه الإمكانية للتخزين الدائم . وبعد ذلك يمكن استرجاع السجلات الفردية من الملف ، وإجراء التعديل عليها ، وكتابة محتوياتها ، ثم تخزينها مرة أخرى . وسوف نرى كيف يمكن تحقيق ذلك فى الفصل القادم . ويجب أن يكون واضحا أن المثال الحالى يهدف إلى توضيح كيفية تشغيل بيانات من نوع السجلات فى البسكال فقط .

## أسئلة للمراجعة Review Questions

(١) ماهو الفرق الأساسى بين السجل والمنظومة ؟

(٢) ماذا يعنى الحقل ؟ وماهى العلاقة بين الحقل والسجل ؟

(٣) لخص قواعد تعريف السجل ؟

(٤) لخص قواعد تعريف متغير من نوع السجل . قارن ذلك بإجابتك للسؤال السابق .

(٥) ماهو نوع البيانات الذى يمكن أن يصاحب عنصر بيانات فرديًا داخل الحقل ؟

(٦) فى أى نوع من أنواع التطبيقات يكون من المرغوب فيه لعنصر سجل أن يصبح منظومة ؟ وضح ذلك .

(٧) فى أى نوع من أنواع التطبيقات يكون من المرغوب فيه لعنصر منظومة أن يكون سجلا ؟ وضح ذلك ، وقارن مع إجابتك للسؤال السابق .

(٨) كيف يمكن ضغط سجل ؟ وماهى عيوب ومميزات استخدام السجلات المضغوطة ؟

(٩) هل يمكن استخدام نفس اسم الحقل فى سجلين مختلفين ؟ وضح ذلك .

(١٠) هل يمكن تحديد محتويات أحد السجلات لسجل آخر ؟ ماهى القيود التى تقع على هذا النوع من التحديد ؟

(١١) ماهى طرق استخدام عناصر السجل داخل برنامج البسكال ؟

(١٢) ماهو المحدد للحقل ؟ وكيف يكتب ؟ وكيف يستخدم ؟

(١٣) افرض أن عنصر فردى من سجل عبارة عن عنصر بيانات مرتب . كيف يمكن الاتصال بعنصر واحد من عنصر البيانات هذا ؟

(١٤) ماهو الغرض من مكون WITH ؟

(١٥) لخص قواعد استخدام مكون WITH. هل يمكن استخدام أسماء سجلات متعددة فى مكون WITH واحد ؟

(١٦) افرض أن عبارة WITH واحدة تحتوى على متغيرات متعددة من نوع السجل ، وبها اسم حقل مشترك . أى سجل من هذه السجلات يتم الاتصال به عندما يتم الاتصال بحقل داخل المكون ؟ وكيف يمكن الاتصال بحقل له نفس الاسم ، لكنه ينتمى إلى سجل آخر ؟

(١٧) اذكر عدة مميزات لاستخدام مكون WITH عندما يكون ذلك مناسبا .

(١٨) ماذا يعنى الجزء المتغير من السجل ؟ وكيف يختلف هذا الجزء عن الجزء الثابت ؟

(١٩) هل يجب أن يوجد فى كل سجل جزء متغير ؟ وهل يجب أن يوجد به جزء ثابت ؟ وهل يمكن أن يكون فى السجل أكثر من جزء متغير واحد ؟

(٢٠) لخص قواعد تعريف سجل به كل من الجزء الثابت والجزء المتغير . ضاهِ تعريف الجزء المتغير مع مكون الحالة CASE الذى سبق ذكره فى الفصل السادس .

(٢١) إذا ما احتوى السجل على جزء متغير ، فأين يجب أن يقع هذا الجزء ؟

(٢٢) ماهو حقل شعار ؟ وماهو الغرض منه ؟

(٢٣) مانوع البيانات التى يمكن أن تصاحب حقل شعار ؟

(٢٤) هل يجب أن يوجد حقل شعار فى كل سجل متغير ؟ وضح ذلك .

(٢٥) كيف يمكن الاتصال بعناصر فردية موجودة داخل الجزء المتغير من السجل ؟ وماهى القيود الموضوعة على الاتصال بمثل هذه الحقول المتغيرة ؟

(٢٦) هل يمكن استخدام مكون WITH مع سجلات من النوع المتغير ؟

(٢٧) كيف يمكن تعريف قائمة حقل متغير فارغة ؟ ولماذا يمكن أن يكون هذا شيئا مرغوبا فيه ؟

(٢٨) كيف يمكن عمل تداخل للأجزاء المتغيرة داخل سجل واحد ؟

(٢٩) لماذا لايسمح بالتغييرات فى الجزء المتغير داخل مكون WITH إذا ماظهر اسم الحقل المتغير فى عبارة WITH ؟

(٣٠) ماهى القيود التى يجب أخذها فى الاعتبار عندما يمرر حقل متغير إلى إجراء أو دالة كمؤشر متغير فعلى ؟

(٣١) هل يمكن تمرير معرف حقل شعار إلى إجراء أو دالة كمؤشر متغير فعلى ؟ وضح ذلك .

(٣٢) علق على استخدام الأجزاء المتغيرة بواسطة المبرمجين المبتدئين .

## مسائل محلولة : solved Problems

(٣٣) فيما يلى عدة توضيحات سجلات

(*a*)
```
VAR sample : RECORD
                first  : 1..100:
                second : real;
                third  : boolean
             END;
```

(*b*)
```
TYPE demo = RECORD
               first  : 1..100:
               second : real;
               third  : boolean
            END;
VAR sample1,sample2 : demo;
```

(*c*)
```
TYPE line = PACKED ARRAY [1..80] OF char;
     personal = RECORD
                   name,street,city : line
                END;
VAR employee : ARRAY [1..500] OF personal;
```

(*d*)
```
TYPE color = (red,green,blue);
     sample = PACKED RECORD
                 first  : color;
                 second : 1..132;
                 third  : char
              END;
VAR trial : sample;
```

(*e*)
```
TYPE color = (red,green,blue);
     sample = RECORD
                 CASE select : color OF
                    red   : (first  : 100..199;
                             second : 200..299);
                    green : (third  : 300..399);
                    blue  : ()
              END;
VAR demo : ARRAY [1..500] OF sample;
```

لاحظ أن sample لايحتوى على جزء ثابت . لاحظ أيضا أن قائمة الحقل المتغير الفارغ تناظر اسم الحالة blue .

```
(f) TYPE color   = (red,green,blue);
         texture = (coarse,fine);
         sample  = PACKED RECORD
                     first  : color;
                     second : 1..132;
                     CASE color OF
                        red   : (first  : 100..199;
                                 second : 200..299);
                        green : ();
                        blue  : (CASE texture OF
                                   coarse : (third : 300..399);
                                   fine   : ())
                   END;
    VAR demo1,demo2 : ARRAY [1..99] OF sample;
```

لاحظ أن الأجزاء المتغيرة المتداخلة فى sample لاتحتوى على معرفات حقل شعار .

(٣٤) فيما يلى تخطيطات هيكلية لاستخدام سجلات وعناصر سجلات فى مواقف برمجة تقليدية .

```
(a) PROGRAM sample(input,output);
    TYPE line = PACKED ARRAY [1..80] OF char;
         personal = RECORD
                      name,street,city : line
                    END;
    VAR employee : personal;
    BEGIN
       .
       .
       (* enter input data *)
       .
       .
       writeln(employee.name);
       writeln(employee.street);
       writeln(employee.city);
       .
       .
    END.
(b) PROGRAM sample(input,output);
    TYPE line = PACKED ARRAY [1..80] OF char;
         personal = RECORD
                      name,street,city : line
                    END;
    VAR employee : ARRAY [1..500] OF personal;
        count : 1..500;
    BEGIN
       .
       .
       (* enter input data *)
       .
       .
       FOR count := 1 TO 500 DO
          BEGIN
             writeln(employee [count].name);
             writeln(employee [count].street);
             writeln(employee [count].city)
          END;
       .
       .
    END.
```

(*c*)
```
PROGRAM sample(input,output);
TYPE line = PACKED ARRAY [1..80] OF char;
     personal = RECORD
                   name,street,city : line
                END;
VAR employee : ARRAY [1..500] OF personal;
    count,total : 1..500;
BEGIN
   .
   .
   (* enter input data *)
   .
   .
   FOR count := 1 TO total DO
      WITH employee [count] DO

         BEGIN
            writeln(employee [count].name);
            writeln(employee [count].street);
            writeln(employee [count].city)
         END;
   .
   .
END.
```

(*d*)
```
PROGRAM sample(input,output);
TYPE line = PACKED ARRAY [1..80] OF char;
     personal = RECORD
                   name,street,city : line
                END;
VAR customer,employee : ARRAY [1..500] OF personal
    i,j,total : 1..500;
BEGIN
   .
   .
   (* enter input data *)
   .
   .
   FOR i := 1 TO total DO
      WITH customer [i] DO
         FOR j := 1 TO total DO
            IF name = employee [j].name THEN
               BEGIN
                  writeln(name);
                  writeln(street);
                  writeln(city)
               END;
   .
   .
END.
```

لاحظ أن عناصر السجل المكتوبة هي جزء في customer [i] ، وليست جزءا في employee [i] .

(*e*)
```
PROGRAM sample(input,output);
TYPE status = (single,married,divorced,widowed);
     line = PACKED ARRAY [1..80] OF char;
     background = RECORD
                     name,address : line;
                     employeeno : 1..9999;
                     CASE maritalstatus : status OF
                        single : ();
                        married : (children : 0..10);
                        divorced,widowed :
                                 (children : 0..10;
                                  remarried : boolean)
                     END;
VAR employees : ARRAY [1..100] OF background;
    count,total : 1..100;
BEGIN
   .
   .
   (* enter input data *)
   .
   .
   FOR count := 1 TO total DO
      WITH employees [count] DO
         BEGIN
            writeln(name);
            writeln(address);
            writeln(employeeno:4);
            IF maritalstatus <> single THEN writeln(children:2);
            IF ((maritalstatus = divorced) OR
                (maritalstatus = widowed)) AND
                (remarried = true) THEN maritalstatus = married
         END;
   .
   .
END.
```

## Supplementary Problems مشاكل متكاملة :

(٣٥) التخطيطات الهيكلية التالية توضح مواقف عديدة مختلفة لاستخدام سجلات وعناصر سجلات ، وبعضها مكتوب بطريقة خاطئة . عرف كل الأخطاء .

(*a*)
```
CONST c1 = 12;
      c2 = 31;
      c3 = 2000;
VAR date : RECORD
              month : 1..c1;
              day   : 1..c2;
              year  : 0..c3
           END;
```

(*b*)
```
TYPE months = (jan,feb,mar,apr,may,jun,
                         jul,aug,sep,oct,nov,dec);
     days   = (sun,mon,tue,wed,thu,fri,sat);
     date   = RECORD
                 name : days;
                 month : months;
                 day : 1..31;
                 year : 0..maxint
              END;
VAR yesterday,today : date;
```

(*c*)
```
TYPE item = RECORD
               sales : real;
               cost : real;
               profit : real;
               bonus : boolean
VAR list : ARRAY [1..1000] OF sales;
```

(*d*)
```
TYPE line = PACKED ARRAY [1..80] OF char;
     nameandaddress = RECORD
                         name,street,city : line
                      END;
           personal = RECORD
                         description : nameandaddress;
                         custno : 1..9999;
                         status : char;
                         balance : real
                      END;
VAR customer : ARRAY [1..10000] OF personal;
```

(*e*)
```
PROGRAM sample(input,output);
TYPE date = RECORD
               month : 1..12;
               day : 1..31;
               year : 1900..2100
            END;
VAR birthday,today : date;
    count : 0..maxint;
BEGIN
   .
   .
   WITH today DO readln(month,day,year);
   .
   .
   count := 0;
   REPEAT
      WITH birthday DO
         readln(month,day,year);
      IF birthday = today THEN count := succ(count);
   UNTIL year = 0;
   writeln(count);
   .
   .
END.
```

(*f*)
```
PROGRAM sample(input,output);
TYPE line = PACKED ARRAY [1..80] OF char;
     date = PACKED RECORD
                 month : 1..12;
                 day : 1..31;
                 year : 1900..2100
              END;
VAR total,count : 1..maxint;
    today : date;
    birthday : RECORD
                   name : line;
                   birthdate : date
               END;
BEGIN
   .
   .
   readln(total);
   WITH today DO readln(month,day,year);
   .
   .
   FOR count := 1 TO total DO
      WITH birthday DO
         BEGIN
            readln(name);
            birthdate := today
         END;
   .
   .
END.
```

(*g*)
```
TYPE item = (stereo,tv);
     name = PACKED ARRAY [1..80] OF char;
     inventory = RECORD
                    stockno : 1..20000;
                    supplier : name;
                    CASE item OF
                       stereo : (power : 1..1000);
                           tv : (tubesize : 1..25;color : char);
                     quantity : integer
                  END;
VAR stockitem,backorder : inventory;
```

(*h*)
```
PROGRAM sample(input,output);
TYPE status = (single,married,divorced,widowed);
     background = RECORD
                     CASE maritalstatus : status OF
                        single : ();
                        married : (children : 0..10);
                        divorced,widowed :
                                (children : 0..10; remarried : boolean)
                  END;
VAR employees : ARRAY [1..100] OF background;
BEGIN
   .
   .
   maritalstatus := single;
   writeln(divorced.children);
   .
   .
END.
```

(*i*)
```
PROGRAM sample(input,output);
TYPE status = (single,married,divorced,widowed);
     background = RECORD
                     CASE maritalstatus : status OF
                        single : ();
                        married : (children : 0..10);
                        divorced,widowed :
                                 (children : 0..10;remarried : boolean)
                  END;
VAR employees : ARRAY [1..100] OF background;
            i : 1..100;

PROCEDURE setup(VAR test : status);
BEGIN
   .
   .
   IF test = widowed THEN test := single
                     ELSE test := succ(test)
   .
   .
END;

BEGIN   (* main action statements *)
   FOR i := 1 TO 100 DO
      WITH employees [i] DO
        BEGIN
           .
           .
           setup(maritalstatus);
           .
           .
        END
END.
```

## Programming Problems — مشاكل برمجة :

(٣٦) عدل برنامج فواتير العملاء الموجود فى مثال (١٠ - ٢١) ، بحيث يمكن كتابة أى تقرير من التقارير التالية :

أ - تقرير حالة كل العملاء ( ينتجه البرنامج نفسه ) .

ب - تقرير حالة العملاء أصحاب الحسابات متأخر ومقصر فقط .

ج - تقرير حالة العملاء أصحاب الحسابات المقصرة فقط .

حدد جزءًا لانتاج قائمة ، عند تنفيذ البرنامج يستطيع أن يختار المستفيد منها التقرير المطلوب إنتاجه . وبعد ذلك يعود البرنامج إلى القائمة بعد طباعة التقارير ، وتكون هناك إمكانية لإنتاج عدة تقارير مختلفة .

(٣٧) عدل برنامج مراقبة المخزون الموجود فى مثال (١٠ ٢٨) ، بحيث يمكن كتابة أى تقرير من التقارير التالية :

أ - تقرير بقائمة كاملة للمخزون ( ينتجه البرنامج نفسه ) .

ب - تقرير بقائمة لكل العناصر التى لها أوامر خلفية .

ج - تقرير بقائمة لكل أجهزة ستريو الموجودة حاليا فى المخزن .

د - تقرير بقائمة لكل أجهزة التليفزيون الملون الموجودة حاليا فى المخزن .

هـ - تقرير بقائمة كل أجهزة التليفزيون الأبيض والأسود الموجودة حاليا فى المخزن .

و - تقرير بقائمة لكل الأجهزة القابلة للنقل الموجودة حاليا فى المخزن .

ابدأ بإنتاج قائمة عندما ينفذ البرنامج ، مع السماح للمستفيد بأن يختار تقريرا معينا ، وبعد ذلك يعود البرنامج إلى هذه القائمة بعد إنتاج كل تقرير ، وتكون هناك إمكانية لإنتاج التقارير المتنوعة .

(٣٨) أعد كتابة كل برنامج من برامج البسكال التالية ، بحيث إنها تستخدم تكوين بيانات من نوع السجلات .

أ - برنامج الاستهلاك المقدم فى مثالى ( ٦ -١٧) ، ( ٧ -١٢) .

ب - البرنامج المعطى فى مثال (٨ - ٩ ) لحساب عدد الأيام التى تقع بين تاريخين محددين .

ج - البرنامج المستخدم فى قراءة وكتابة قائمة بالأسماء والعناوين الموجود فى مثال ( ٩ - ٧ ) .

(٣٩) عدل برنامج البسكال الموجود فى مثال ( ٩ - ٧ ) ، بحيث يمكنه أن يقرأ عدة أسماء وعناوين مختلفة داخل الكمبيوتر ، ويعيد ترتيبها ترتيبا أبجديا ، ثم يكتبها بترتيبها الجديد . حدد تكوين بيانات من نوع السجل داخل البرنامج .

(٤٠) عدل البرنامج الذى ينتج pig latin الموجود فى مثال ( ٩ -٢٧) ، بحيث إنه يقبل أسطر متعددة من النص . مثل كل سطر من أسطر النص بأنه سجل منفصل مع وجود ثلاثة حقول داخل كل سجل ، وهى .

أ - السطر الأصلى فى النص .

ب - عدد الكلمات الموجودة فى السطر .

ج - سطر النص المعدل ( أى السطر مكتوبا باستخدام pig latin ) .

( حدد الزيادات الموجودة فى المشكلة رقم ( ٥٠ من الفصل التاسع ) . أدخل علامات التنقيط . والحروف الكبيرة ، وأصوات الحروف المزدوجة )

(٤١) اكتب برنامجا كاملا بلغة البسكال لكل مشكلة من مشاكل البرمجة التالية ، مستخدما تكوينات بيانات من نوع السـجل .

أ – مشكلة تحديد متوسط درجات الكلية الموجودة فى المشكلة رقم ( ٥٢ من الفصل التاسع ) الجزء (أ) .

ب – الصيغة الأخرى لمشكلة تحديد متوسط درجات الكلية الموجودة فى المشكلة رقم ( ٥٢ من الفصل التاسع ) الجزء (د) .

ج – المشكلة التى تجرى توافقا بين أسماء البلاد والعواصم المناظرة لها ( انظر المشكلة رقم ( ٥٦ من الفصل التاسع ) .

(٤٢) اكتب برنامجا كاملا بلغة البسكال ، يقبل المعلومات التالية لكل فريق من فرق البيسبول base ball و فرق كرة القدم .

١ – اسم الفريق ، بما فى ذلك المدينة .

٢ – عدد مرات الكسب .

٣ – عدد مرات الخسارة .

وبالنسبة لفرق البيسبول تضاف المعلومات التالية :

١ – عدد الضربات ( فى كل الموسم ) hits .

٢ – عدد الدورات runs .

٣ – عدد الأخطاء .

٤ – عدد المباريات extra - inning .

وبالمثل أضف لفريق كرة القدم المعلومات التالية :

١ – عدد مرات الإيقاف tics .

٢ – عدد مرات اللمس touchdowns .

٣ – عدد الأجوال field goals .

٤ – إجمالى المكسب ( على مدى الموسم ) total yards gined .

٥ – إجمالى الخسارة total yards given up to oppancnts .

أدخل هذه المعلومات الخاصة بكل الفرق ، ثم أعد ترتيب الفرق ، واطبع قائمة بها طبقا لسجلات المكسب والخسـارة ، مستخدما أسلوب إعادة الترتيب المستخدم فى مثال ( ٩ – ٨ ) .

تأكد من استخدام تكوين بيانات من نوع السجل ، تشمل الجزء الثابت والجزء المتغير من السجل . ويجب أن تحفظ هذه السجلات فى منظومة .

اختبر البرنامج ، مستخدما إحصاءات حديثة لهذه الغرض .

## الفصل الحادى عشر

# الملـفات

# Files

نركز انتباهنا الآن على تكوين هام آخر للبيانات وهو الملف . الملف file مثل المنظومة عبارة عن تجميع لعناصر بيانات كلها من نفس النوع . وعلى عكس المنظومة ، يمكن تخزين الملف فى إحدى وحدات التخزين المساعد ( أى وحدة تشغيل شريط ، أو وحدة تشغيل قرص ) . وعلى هذا ... فيسمح لنا تكون الملف بالتخزين الدائم للمعلومات والاتصال بهذه المعلومات عند الحاجة لها . ويتطلب العديد من التطبيقات الهامة هذه الإمكانية .

## 1. PRELIMINARIES ١ – أساسيات

بصفة عامة يوجد نوعان من الملفات يمكن إنتاجها بواسطة برنامج الكمبيوتر ، وكذلك الاتصال بهما ، وهما الملفات الدائمة والملفات المؤقتة . وتحفظ الملفات الدائمة permanent files فى إحدى وحدات تشغيل التخزين المساعد . ويمكن الاتصال بمحتويات هذه الملفات وتعديلها ، إما بواسطة البرنامج الذى أنتجها ، أو بواسطة برنامج آخر ، وذلك فى أى وقت . ويشار فى البسكال إلى الملفات الدائمة بأنها ملفات خارجية external files .

وتخزين الملفات المؤقتة temporary files داخل الذاكرة الرئيسية للكمبيوتر . ويفقد الملف المؤقت بمجرد الانتهاء من تنفيذ البرنامج الذى أنتجه . وعلى هذا ... فالملفات المؤقتة أقل نفعا ، أو أقل استخداما من الملفات الدائمة . ويشار فى البسكال إلى الملفات المؤقتة بأنها ملفات داخلية internal files .

ويمكن تنظيم كل الملفات ، سواء أكانت دائمة أم مؤقتة بإحدى طريقتين ، إما تتابعيا ، أو عشوائيا ( الطريقة الأخيرة تعرف بأن الملف هو ملف اتصال مباشر direct - access وهو اسم أكثر تعبيرا عن تنظيم الملف ) . فى الملف التتابعى Sequential File تخزن مكونات الملف متتابعة واحدة تلو الأخرى . والوصول إلى إحدى مكونات الملف يكون من الضرورى البدء ببداية الملف ، والبحث خلال الملف حتى الوصول إلى المطلوب . وهذا النوع من أنواع الاتصال بطئ ، خاصة إذا ماكان الملف كبيرا . وإنتاج الملفات التتابعية سهل نسبيا ، وهى الملفات الوحيدة التى يمكن استخدامها مع بعض أنواع أوساط التخزين ، وهى الشريط المغناطيسى .

أما فى ملفات الاتصال العشوائى rendom - access file ، فيمكن الاتصال باحدى مكونات الملف مباشرة ، دون البحث خلال محتويات الملف منذ بدايته . ومثل هذه الملفات تيسر اتصالا أسرع كثيرا لإحدى مكونات الملف عن الملفات التتابعية ، وذلك بالرغم من أنه من الصعب جدا إنتاج هذه الملفات وصيانتها .

وبسكال ISO القياسى يستخدم الملفات التتابعية فقط ، وذلك بالرغم من أن بعض مترجمات اللغة تستخدم ملفات الاتصال العشوائى . وسوف نناقش الملفات التتابعية ( القياسية ) فقط فى هذا الكتاب .

## 2. DEFINING A FILE ٢ – تعريف الملف :

أسهل طريقة لتعريف الملف هى اعتبار تعريف الملف كجزء من توضيح متغير له نوع الملف .

والصيغة العامة لمثل هذا التعريف هى :

```
VAR file name : FILE OF type
```

حيث filé name هو معرف يمثل اسم الملف . وتشير type إلى نوع البيانات الخاصة بمكونات الملف الفردية . ويجب أن تكون جميع هذه المكونات من نفس النوع . ويمكن أن تكون من أى نوع ، سواء أكان بسيطا أم مركبا ، فيما عدا أنها لايمكن أن يكون لها نوع ملف . وعلى هذا ... فلا يمكن أن يحتوى الملف ملفا آخر .

## مثال (١١-١)

فيما يلى تعريفًا بسيطًا لملف . وفى هذا التعريف مكونات الملف عبارة عن عناصر بيانات بسيطة من النوع الحرفى .

```
VAR symbols : FILE OF char;
```

لاحظ أن اسم الملف هو Symbol .

## مثال (١١-٢)

اعتبر الآن تعريف ملف ، كل مكون من مكوناته عبارة عن منظومة ذات بعدين .

```
TYPE table = ARRAY [1..50,1..20] OF real;
VAR data : FILE OF table;
```

فى هذه الحالة اسم الملف هو data .

## مثال (١١-٣)

فيما يلى مثالا لتعريف ملف ، كل مكون من مكوناته عبارة عن سجل .

```
TYPE status = (current,overdue,delinquent);
     account = RECORD
                  custname : PACKED ARRAY [1..80] OF char;
                  custno : 1..9999;
                  custtype : status;
                  custbalance : real
               END;
VAR customers : FILE OF account;
```

لاحظ أن كل سجل داخل الملف customers يحتوى على منظومة مضغوطة من النوع الحرفى ( أى متغير سلسلة ) وعلى كمية عددية صحيحة ، ونوع بيانات متعدد ، وكمية عددية حقيقية .

أعتبر الآن طريقة أكثر عمومية لتعريف الملفات ، حيث نعرف فيها نوع الملف أولا ، ثم توضيح متغيرا أو أكثر ، ليكون من نفس هذا النوع للملف . ( سبق أن استخدمنا هذه الطريقة فى تعريف منظومات وسجلات ) . والصيغة العامة من هذا النوع لتعريف الملف ، هى :

```
TYPE name = FILE OF type
```

حيث تشير type إلى نوع بيانات كل مكون من مكونات الملف . ويجب أن يكون هذا النوع سبق تعريفه .

## مثـال (١١-٤)

فيما يلى مثالا لتعريف ملف ، موجود فيه ملفان مختلفان ، لهما مكونات من السجلات التى لها نفس النوع .

```
TYPE status = (current,overdue,delinquent);
     account = RECORD
                 custname : PACKED ARRAY [1..80] OF char;
                 custno : 1..9999;
                 custtype : status;
                 custbalance : real
              END;
     customers = FILE OF account;
VAR oldcustomers,newcustomers : customers;
```

وعلى هذا .. فإن كلا من newcustomers, oldcustomers ملفا من نوع customers . ومكونات كل ملف هى سجلات من نوع account كما هو معرف أعلاه .

لاحظ أن طول الملف لايحدد على الإطلاق كجزء من تعريف الملف . ويختلف هذا الموقف عن أنواع البيانات المرتبة الأخرى ، حيث تحدد عدد المكونات بوضوح ( فى حالة المنظومات ) أو ضمنيا ( فى حالة السجلات ) ، وذلك داخل تعريف نوع البيانات . وكقاعدة عامة ... يتحدد أقصى طول للملف بواسطة السعة الطبيعية للوسط الذى يخزن عليه الملف .

يجب أن تمرر الملفات الخارجية إلى برنامج البسكال كمؤشرات ، ويتحقق ذلك بوضع أسماء الملفات الخارجية فى عنوان البرنامج ، محصورة بين قوسين . وإذا كان هناك أكثر من ملف واحد ، فتنفصل الملفات عن بعضها بواسطة فواصل . مواصفات المؤشرات هذه توجد بالإضافة إلى تعريفات الملف المطلوبة والتى يجب أن توجد داخل المجموعة الرئيسية من البرنامج .

## مثـال (١١-٥)

افرض أن أحد برامج البسكال يستخدم ملفًا خارجيًا اسمه customers ، والموصوف فى مثال (١١- ٣ ) . فيما يلى تخطيطا هيكليًا لهذا البرنامج :

```
PROGRAM sample(customers);
   .
   .
TYPE status = (current,overdue,delinquent);
     account = RECORD
                 custname : PACKED ARRAY [1..80] OF char;
                 custno : 1..9999;
                 custtype : status;
                 custbalance : real
              END;
```

```
VAR customers : FILE OF account;
    .
    .
BEGIN   (* main action statements *)
    .
    .
    (* process. the records within customers  *)
    .
    .
END.
```

لاحظ أن اسم الملف customers موجود فى عنوان البرنامج بين قوسين . ويتبع ذلك تعريف الملف ، والذى يظهر داخل جزء التوضيح من المجموعة الأساسية للبرنامج .

ويوجد فى البسكال ملفان ( سبق توضيحهما ) قياسيان ، هما : output , input ، وعن طريقهما يتم توجيه معلومات من النوع الحرفى من وحدة مدخلات ، وإلى وحدة مخرجات على التوالى . يجب أن يتواجد اسما الملفين هذين فى عنوان البرنامج ، إذا ماكان البرنامج سيقوم بالاتصال بهما . وحيث إنهما من معالم اللغة القياسية ، فهما ليسا على أية حال فى حاجة لأن يعرفا بوضوح داخل البرنامج . وفى واقع الأمر ... إننا استخدمنا هذين الملفين القياسيين خلال هذا الكتاب ، وسوف نقول المزيد عن ملفات المدخلات والمخرجات سابقة التوضيح فى قسم ( ٨ من هذا الفصل ) .

## مثـال (١١-٦)

افرض الآن أن برنامج البسكال الموضح تخطيطه الهيكلى فى مثال (١١- ٥ ) تم تعديله ليستخدم ملفات output , input قياسية . فيما يلى تخطيطا مناسبا للبرنامج .

```
PROGRAM sample(input,output,customers);
    .
    .
TYPE status = (current,overdue,delinquent);
     account = RECORD
                  custname : PACKED ARRAY [1..80] OF char;
                  custno : 1..9999;
                  custtype : status;
                  custbalance : real
               END;
VAR customers : FILE OF account;
    .
    .
BEGIN   (* main action statements *)
    .
    .
   (* process the records within customers  *)
    .
    .
END.
```

وهذا التخطيط متطابق مع نظيره الموجود فى مثال (١١- ٥ ) ، باستثناء وجود اسمى ملفى output , input كمؤشرين فى عنوان البرنامج .

ويجب أن يكون واضحا أنه ليس هناك حاجة لظهور ملفى output , input معا دائما فى عنوان البرنامج . فيجب تحديد الملفات المطلوب وجودها فعلا فقط . فإذا ماكان البرنامج ينتج مخرجات ، دون الحاجة إلى بيانات مدخـلات ، فيجب أن يحتوى عنوان البرنامج على اسم ملف بيانات output ، ولاحاجة إلى اسم ملف input .

## ٣ - إنتاج الملف 3. CREATING A FILE

نركز انتباهنا الآن إلى إنتاج ملف فى البسكال . يجب أن نرى أولا كيفية الاتصال بعنصر بيانات فردى داخل الملف . ويتحقق ذلك عن طريق احتياطى الملف file buffer ، والذى يسمح بقيمة ( أو قيم ) تصاحب مكونات ملف فردى لتنقل بين الملف وبرنامجه المضيف .

وفى واقع الأمر فإن احتياطى الملف file buffer هو نوع خاص من أنواع المتغيرات ، والتى تعرف تلقائيا عندما يتم تعريف الملف المناظر . ويجب أن يكون لاحتياطى الملف نفس اسم الملف ويكون له سهم رأسى مضاف فى نهايته . وعلى هذا ... فإذا ماكان اسم الملف file name يمثل اسم الملف ، فإن احتياطى الملف المناظر يسمى file name . ويجب أن يكون نوع احتياطى الملف هو نفس نوع مكونات الملف .

### مثـال (١١-٧)

افرض أن أحد برامج البسكال يحتوى على تعريف ملف اسمه data ، وعلى هذا ... فإن احتياطى الملف المناظر يسمى data ، فإذا ماكانت مكونات الملف سجلات ، فإن data يمثل مثل هذه السجلات ، وعند إنتاج ملف ، فإن الفكرة الأساسية هى تحديد عناصر البيانات التى تُكون أحد مكونات ملف لاحتياطى الملف ، وبعد ذلك تنقل هذه القيم من احتياطى الملف إلى الملف نفسه . ويتم إنتاج محتويات الملف بتكرار هذه العملية لكل مكون من مكونات الملف .

وأول خطوة فى إنتاج الملف هى إعداد الملف للكتابة . ويتم تحقيق ذلك باستخدام إجراء قياسى اسمه rewrite ، وعلى هذا فإننا نبدأ بكتابة مايلى :

```
rewrite(file name);
```

فإذا ماكان إنتاج الملف المحدد يحدث لأول مرة ، فإن عبارة rewrite تحدد ببساطة بداية الملف . أما اذا ماكان الملف موجودا بالفعل بنفس الاسم المعطى ، فإن تأثير عبارة rewrite ذلك هو حذف كل المعلومات الموجودة داخل الملف مثلا ، ثم تحديد بداية للملف الجديد .

وتحدد عناصر البيانات التى يتكون فيها مكونات ملف فردى لاحتياطى الملف بنفس الطريقة التى يتم تحديد عناصر بيانات بها للمتغيرات التى تناظرها . فإذا مااحتوت المكونات على عناصر بيانات من النوع البسيط ، فإن كل تحديد لاحتياطى الملف يحتوى على قيمة فردية . ويجب بالطبع أن تكون هذه القيم متوافقة مع نوع احتياطـــى الملـــف . ( عادة ماتكون عناصر البيانات من نفس نوع احتياطى الملف ) .

إذا مااحتوى الملف على مكونات من النوع المرتب ( أى على منظومات أو سجلات ) ؛ فيكون احتياطى الملف متغيرا مناظرا من النوع المرتب ، وعلى هذا ... يمكن تحديد قيمة المتغير المناظر من النوع المرتب للاحتياطى ، أو يمكن تحديد عنصر بيانات فردى منفصلا . وفى الحالة الأخيرة فإن التحديدات الفردية لمكونات الاحتياطى يجب أن تتبع القواعد المعتادة لتحديد عناصر البيانات لأنواع البيانات المرتبة .

بعد تحديد عناصر البيانات اللازمة لاحتياطى الملف ، تنقل هذه المعلومات إلى الملف بواسطة إجراء قياســى اسمه Put . وعلى هذا يمكننا كتابة مايلى :

```
file name↑ := . . . ;
put(file name);
```

فإذا ماتكرر تنفيذ هاتين العبارتين ؛ فينتج عن ذلك كتابة تسلسل من عناصر البيانات فى الملف . تخزن عناصر البيانات هذه داخل الملف بنفس الترتيب الذى كتبت به . وفى كلمات أخرى ... فإن عناصر البيانات المصاحبة للمكون الحالى توضع فى نهاية الملف ، تالية لعناصر البيانات ( قيم مكونات ) التى سبق كتابتها من قبل .

## مثـال (١١-٨)

فيما يلى مثالا لبرنامج بسكال ينتج ملفا اسمه data . يحتوى هذا الملف على مكونات من النوع البسيط ( أعداد حقيقية موجبة ) .

```
PROGRAM createfile1(input,output,data);
VAR data : FILE OF real;
    item : real;
    count,total : 1..maxint;
BEGIN
   write('How many values will be entered? ');
   readln(total);
   rewrite(data);
   FOR count := 1 TO total DO
      BEGIN
         readln(item);
         data↑ := item;
         put(data)
      END
END.
```

لاحظ أن هذا البرنامج يتسبب فى إدخال بيانات من وحدة مدخلات ( أى لوحة مفاتيح ) ، ثم يكتب هذه البيانات على ملف جديد اسمه data .

ويبدأ البرنامج بسؤال المستفيد عن عدد القيم ( أى عدد مكونات الملف ) التى يراد إدخالها . وبعد قراءة هذه القيمة داخل الكمبيوتر ، يتم إعداد الملف الجديد عن طريق عبارة rewrite ، وبعد ذلك يستخدم البرنامج دورة FOR - TO لقراءة قيمة كل مكون من مكونات الملف من وحدة المدخلات ، ثم كتابتها فى الملف .

لاحظ أن هذا البرنامج يتطلب أن يعرف المستفيد مسبقا عدد عناصر البيانات ( أى عدد مكونات الملف ) التى سوف تضاف الى الملف . وهذا غير عملى بعض الشئ ، لكنه يكون عمليا إذا ماكان الملف طويلا نسبيا . والمثال التالى يقدم طريقة أفضل .

## مثـال (١١-٩)

فيما يلى صيغة أخرى للبرنامج الموجود فى مثال (١١- ٨ ) . يمثل هذا البرنامج تعديلا للصيغة السابقة ، بمعنى أن عدد عناصر البيانات ليس هناك حاجة لمعرفته مسبقا . ويستمر المستفيد ببساطة فى إدخال البيانات حتى تظهر قيمة سالبة تحدد إنتهاء الملف ( لاحظ أن هذا المخطط يوجد عليه قيد ، وهو أن مكونات الملف الفعلية يجب ألا تكون سالبة ) .

```
PROGRAM createfile2(input,data);
VAR data : FILE OF real;
    item : real;
BEGIN
```

(تكملة البرنامج فى الصفحة التالية)

```
        rewrite(data);
        readln(item);
        WHILE item >= 0 DO
           BEGIN
              data↑ := item;
              put(data);
              readln(item)
           END
     END.
```

لاحظ أن عنوان البرنامج يحتوى على مؤشرات ملفات output , input . ولايحتاج هذا البرنامج الى استخدام ملف output قياسى ، وذلك لعدم وجود ملقنات يكتبها البرنامج . ( قارن ذلك بالبرنامج الموجود فى مثال١١-٨ ، والذى ينتج ملقنات لإجمالى عدد عناصر البيانات ) .

إذا ماكانت مكونات الملف مرتبة ، فقد يكون من المرغوب فيه إدخال عنصر بيانات فردى لكل مكون من مكونات الملف بواسطة إجراء . وفى مثل هذه الحالات يجب أن يمر متغير مرتب من نفس نوع احتياطى الملف إلى الإجراء كمؤشر فعلى . ويكون المؤشر الرسمى المناظر مؤشراً متغيراً من نفس نوع احتياطى الملف أيضا . وبعد الاتصال بالإجراء يجب أن تحدد البيانات التى يمثلها المؤشر الرسمى لاحتياطى الملف ، مع السماح بكتابة مكونات الملف فى الملف . والتفاصيل موضحة فى المثال التالى :

## مثـال (١١-١٠)

إنتاج ملف لنظام فواتير العملاء . اعتبر الآن برنامجا ينتج ملفا مكونا من سجلات فواتير عملاء كما هو معرف فى مثال (١١- ٥ ) . يظهر توضيح البرنامج على النحو التالى :

```
TYPE status = (current,overdue,delinquent);
     account = RECORD
                  custname : PACKED ARRAY [1..80] OF char;
                  custno : 1..9999;
                  custtype : status;
                  custbalance : real
               END;
VAR customeracct : account;
    customers : FILE OF account;
    count : 0..80;
```

هذه التوضيحات هى نفسها مثل التوضيحات فى مثال (١١- ٥ ) ، فيما عدا توضيحات متغير السجل الإضافى customeracct ، والذى يستخدم كمؤشر ، والمتغير صحيح النوع count .

افرض أننا نستخدم إجراء فى إدخال عناصر البيانات الفردية لكل مكون من مكونات الملف ( فى هذه الحالة لكل سجل ) . سوف يقرأ الإجراء قيم اسم العميل ، ورقم العميل ، وموازنة العميل فى الكمبيوتر . ويمكن عند ذلك تحديد نوع العميل ( حالته status ) طبقا لقيمة موازنة العميل ( أو لأى معيار آخر ) .

ويمكن كتابة إجراء المدخلات input على النحو التالى :

```
PROCEDURE readinput(VAR info : account);
(* Enter input data for one record *)
BEGIN
   WITH info DO
      BEGIN
         write('Customer number: ');
         readln(custno);
         IF custno <> 9999 THEN
            BEGIN
               write('Name: ');
               count := 0;
               WHILE NOT eoln DO
                  BEGIN
                     count := count + 1;
                     read(custname[count])
                  END;
               writeln;
               write('Current balance: ');
               readln(custbalance);
               writeln;
               custtype := current
            END
      END   (* WITH info *)
END;   (* readinput *)
```

لاحظ أن info هو مؤشر رسمى من نفس نوع account . وعلى هذا ... فإن info متغير من نوع السجل . لاحظ أيضا أن الإجراء يقرأ اسم العميل أولا ، ثم يستمر إذا ماكان رقم العميل لايساوى 9999 فقط . وعلى هذا ... فإن رقم العميل 9999 يقدم طريقة مقنعة لإيقاف العمل .

والجزء الرئيسى للبرنامج بسيط ، حيث يتم إعداد الملف الجديد customers ، ثم قراءة أول سجل ، وبعد ذلك يتم تكرار الخطوات التالية ، طالما أن رقم العميل لايساوى 9999 .

١ – نحدد أحدث البيانات ( أحدث سجل ) لاحتياطى الملف .

٢ – نكتب احتياطى الملف ( أحدث سجل ) فى الملف .

٣ – نقرأ السجل الجديد .

وفيما يلى برنامجا كاملا لأداء ذلك .

```
PROGRAM createfile3(input,output,customers);
(* THIS PROGRAM CREATES A FILE OF CUSTOMER RECORDS *)
TYPE status = (current,overdue,delinquent);
     account = RECORD
                 custname : PACKED ARRAY [1..80] OF char;
                 custno : 1..9999;
                 custtype : status;
                 custbalance : real
              END;
```

(تكملة البرنامج فى الصفحة التالية)

```
VAR customeracct : account;
    customers : FILE OF account;
    count : 0..80;

PROCEDURE readinput(VAR info : account);
(* Enter input data for one record *)
BEGIN
   WITH info DO
      BEGIN
         write('Customer number: ');
         readln(custno);
         IF custno <> 9999 THEN
            BEGIN
               write('Name: ');
               count := 0;
               WHILE NOT eoln DO
                  BEGIN
                     count := count + 1;
                     read(custname[count])
                  END;
               writeln;
               write('Current balance: ');
               readln(custbalance);
               writeln;
               custtype := current
            END
      END   (* WITH info *)
END;   (* readinput *)

BEGIN   (* main action statements *)
   rewrite(customers);
   readinput(customeracct);
   WHILE customeracct.custno < 9999 DO
      BEGIN
         customers↑ := customeracct;
         put(customers);
         readinput(customeracct)
      END
END.
```

## ٤ – المزيد عن عبارة WRITE : 4. MORE ABOUT THE WRITE STATEMENT

من الأمثلة السابقة رأينا أن العبارات :

```
file name↑ := identifier;
put(file name);
```

عادة ماتستخدم فى كتابة مكونات ملف فى الملف . ويمكن تحقيق نفس الشئ بسهولة أكثر ، وذلك بكتابة :

```
write(file name, identifier);
```

حيث يمثل identifier متغيرا أو أى معرف تحدد قيمته ( أو قيمة فى حالة احتياطى ملف من النوع المرتب ) لاحتياطى الملف . ولايسمح هذه البديل لنا باستبدال عبارتين بعبارة واحد فقط ، بل إنه يلغى أيضا الحاجة لأى إشارة صريحة لاحتياطى الملف .

## مثـال (١١-١١)

أعتبر مرة أخرى برنامج البسكال البسيط الموضح فى مثال (١١- ٩ ) . يمكن كتابة هذا البرنامج على النحو التالى أيضا :

```
PROGRAM createfile4(input,data);
VAR data : FILE OF real;
    item : real;
BEGIN
   rewrite(data);
   readln(item);
   WHILE item >= 0 DO
      BEGIN
         write(data,item);
         readln(item)
      END
END.
```

لاحظ أننا استبدلنا العبارتين :

```
data↑ := item;
put(data);
```

فى المثال السابق بعبارة write واحدة ، وهى :

```
write(data,item);
```

## مثـال (١١-١٢)

دعنا نعود الآن إلى برنامج نظام الفواتير الموجود فى مثال (١١ - ١٠) . يمكن تعديل هذا البرنامج ليأخذ الشـكل التالى :

```
PROGRAM createfile5(input,output,customers);

(* THIS PROGRAM CREATES A FILE OF CUSTOMER ACCOUNTS *)

TYPE status = (current,overdue,delinquent);
     account = RECORD
                  custname : PACKED ARRAY [1..80] OF char;
                  custno : 1..9999;
                  custtype : status;
                  custbalance : real
               END;
VAR customeracct : account;
    customers : FILE OF account;
    count : 0..80;

PROCEDURE readinput(VAR info : account);
(* Enter input data for one record *)
BEGIN
   WITH info DO
      BEGIN
```

(تكملة البرنامج فى الصفحة التالية)

```
            write('Customer number: ');
            readln(custno);
            IF custno <> 9999 THEN
               BEGIN
                  write('Name: ');
                  count := 0;
                  WHILE NOT eoln DO
                     BEGIN
                        count := count + 1;
                        read(custname[count])
                     END;
                  writeln;
                  write('Current balance: ');
                  readln(custbalance);
                  writeln;
                  custtype := current
               END
      END   (* WITH info *)
END;   (* readinput *)

BEGIN   (* main action statements *)
   rewrite(customers);
   readinput(customeracct);
   WHILE customeracct.custno < 9999 DO
      BEGIN
         write(customers,customeracct);
         readinput(customeracct)
      END
END.
```

استبدلنا فى هذه الحالة العبارتين :

```
customers↑ := customeracct;
put(customers);
```

الموجودتين فى المجموعة الرئيسية للبرنامج بعبارة write واحدة ، وهى :

```
write(customers,customeracct);
```

ويمكن أن تحتوى عبارة write على معرفات متعددة . إذا كانت هناك رغبة فى ذلك فى هذه الحالة تظهر عبارة write فى الصورة التالية :

write(*file name*, *identifier 1*, *identifier 2*, . . . , *identifier n*);

والتى تكافئ :

```
BEGIN
   write(file name, identifier 1);
   write(file name, identifier 2);
      .
      .
      .
   write(file name, identifier n)
END;
```

ويجب أن يكون مفهوما على أية حال أن نوع البيانات الذى يصاحب كل معرف يجب أن يكون من نفس نوع احتياطى الملف ، أو متوافقًا معه .

أكثر من ذلك ... فالعناصر المسرودة فى عبارة write ليس هناك حاجة لأنها تكون معرفات فردية . ويمكن أن تكون تعبيرات أيضا ، على أن تكون أنواع البيانات الناتجة منها متوافقة مع نفس نوع احتياطى الملف .

وعلى هذا ... فإن الصيغة العامة لعبارة write يمكن كتابتها على النحو التالى :

write(*file name*, *output item 1*, *output item 2*, . . . , *output item n*);

## مثال (١١-١٣)

فيما يلى برنامج بسكال بسيطًا ، ينتج ملفا يحتوى على أول ١٠٠ عدد صحيح ومربعاتها وجذورها التربيعية .

```
PROGRAM createfile6(data);

(* GENERATE A FILE CONTAINING THE FIRST 100 INTEGERS,
                  THEIR SQUARES AND THEIR SQUARE ROOTS *)

VAR data : FILE OF real;
    value : real;
    count : 1..100;

BEGIN   (* main action statements *)
   rewrite(data);
   FOR count := 1 TO 100 DO
      BEGIN
         value := count;
         write(data,value,sqr(value),sqrt(value))
      END
END.
```

لاحظ أن عناصر البيانات مخزنة على هيئة كميات حقيقية . لاحظ أيضا أن كل عدد صحيح يتبعه مربعه وجذره التربيعى . وأخيرا لاحظ أن البرنامج لايستخدم أى من الملفين القياسيين input أو output .

## ٥ – قراءة الملف : 5. READING A FILE

عملية قراءة ملف هى بالضرورة صورة فى المرآة لعملية إنتاج الملف . أول خطوة هى إعداد الملف للقراءة باستخدام إجراء قياسى اسمه reset . ويتم تحقيق ذلك بكتابة مايلى :

reset(*file name*);

إذا كان الملف يحتوى على مكون واحد ، أو أكثر من مكون واحد ، فتتسبب عبارة reset فى قراءة قيمة أول مكون من مكونات الملف ، وتحديدها لاحتياطى الملف . بالإضافة إلى ذلك ... تكون دالة بوليان القياسية eof ( نهاية الملف ) خاطئة false . فإذا كان الملف فارغا – على أية حال – فإن احتياطى الملف يظهر غير معرف ، وتكون قيمة eof حقيقية true .

وبعد قراءة إحدى مكونات الملف وتحديدها لاحتياطى الملف ، يمكن تشغيل محتويات احتياطى الملف إذا كانت هناك رغبة فى ذلك . فمثلا يمكن تحديد مكونات احتياطى الملف لمتغير ، أو طباعة قيمتها ، أو كتابتها فى ملف .. الخ . بعد ذلك يمكن الحصول على المكون الثانى للملف باستخدام إجراء قياسى أسمه get . وعلى هذا ... يحتوى البرنامج بصورة تقليدية على تتابع التعليمات التالية :

```
(* statement to process the contents of file name↑ *);
    .
    .
    .
get(file name);
```

( لاحظ أن أولى التعليمات ممثلة بواسطة تعليق ، حيث إنها ليست معرفة بوضوح ، أى أن طبيعتها الدقيقة تتغير من برنامج لبرنامج آخر ) . وتتكرر هذه التعليمات بصورة تقليدية حتى يتم قراءة وتشغيل كل مكونات الملف ، وعند ذلك تصبح قيمة الدالة eof صحيحة true . وطريقة تحقيق ذلك موضحة فى المثال التالى :

## مثـال (١١-١٤)

فيما يلى برنامج بسكال بسيطًا يقرأ ملفا يحتوى على أعداد حقيقية . ويكتب كل عدد فى وحدة مخرجات قياسية كما يقرأ من الملف .

( لاحظ أنه يمكن استخدام هذا البرنامج فى تشغيل الملفات التى تم إنتاجها فى مثالى (٨ – ١١) و (٩ – ١١) ) .

```
PROGRAM readfile1(output,data);

(* THIS PROGRAM READS REAL NUMBERS FROM A DATA FILE
   AND WRITES THEM TO A STANDARD OUTPUT DEVICE *)

VAR data : FILE OF real;
   item : real;

BEGIN
   reset(data);
   WHILE NOT eof(data) DO
      BEGIN
         item := data↑;
         write(item);
         get(data)
      END
END.
```

هناك عدة ملاحظات فى هذا المثال . لاحظ أولا أن عنوان البرنامج يحتوى على مؤشرات ملف data , output ، ولايحتوى على مؤشر input ( حيث إنه لايوجد نقل معلومات من وحدة مدخلات قياسية ) . لاحظ ثانيا طريقة استخدام المعرفات القياسية get , eof , reset . يحتوى كل معرف على بيانات اسم ملف محصورة بين قوسين . ولاحظ أخيرا أن كل مكون من مكون الملف يتم تشغيله بتحديد محتويات احتياطى الملف أولا لمتغير من النوع الحقيقى ، ثم بعد ذلك كتابة قيمة المتغير فى وحدة المخرجات القياسية .

ويجب أن يكون مفهوما أنه ليس هناك حاجة لمعرفة إجمالى عدد مكونات الملف معرفة مسبقة . وسوف تقرأ المكونات على التوالى بدءًا ببداية الملف ، وتستمر حتى انتهاء الملف . وهذه الطريقة شائعة جدا فى تشغيل محتويات الملف التتابعى .

## 6. MORE ABOUT THE READ STATEMENT : READ ٦ – المزيد عن عبارة

عند القراءة من ملف تتابعى ، فمن الشائع تحديد محتويات احتياطى الملف لمتغير من نوع مناسب ، ثم الحصول على المكون التالى للملف ( أى تحديد محتويات المكون التالى للملف لاحتياطى الملف ) .

يمكن كتابة هاتين العمليتين على النحو التالى :

```
variable name := file name↑ ;
get(file name);
```

وحيث إنه يتكرر استخدام هاتين العبارتين ، فإن لغة البسكال تقدم طريقة أسهل لتحقيق نفس الشئ ، وذلك باستخدام عبارة read . وعلى هذا ... فيمكننا كتابة مايلى :

```
read(file name, variable name);
```

بدلا من العبارتين السابقتين . لاحظ أن احتياطى الملف لايظهر فى عبارة read .

### مثـال (١١-١٥)

فيما يلى صيغة أخرى للبرنامج الموجود فى مثال (١١- ٤ ) . هذه الصيغة تستخدم عبارة read ، بدلا من استخدام get يسبقها عبارة تحديد .

```
PROGRAM readfile2(output,data);
(* THIS PROGRAM READS REAL NUMBERS FROM A DATA FILE
   AND WRITES THEM TO A STANDARD OUTPUT DEVICE *)
VAR data : FILE OF real;
   item : real;
BEGIN
   reset(data);
   WHILE NOT eof(data) DO
      BEGIN
         read(data,item);
         writeln(item)
      END
END.
```

داخل دورة WHILE - DO تؤدى عبارة read مايلى :

١ – تحديد القيمة الحالية لاحتياطى الملف للمتغير الحقيقى item .

٢ – قراءة المكون التالى للملف وتحديده لاحتياطى الملف .

وعند ذلك تتسبب عبارة write فى كتابة القيمة الحالية للمتغير item فى وحدة مخرجات قياسية ( أى عن طريق طابع الأسطر ، أو نهاية طرفية للمشاركة الزمنية ) .

يمكن أن توجد متغيرات متعددة فى عبارة read إذا كانت هناك رغبة فى ذلك . وعلى هذا ... فيمكن كتابة عبارة read على النحو التالى :

```
read(file name, variable name 1, variable name 2, . . . , variable name n);
```

والذى يكافئ مايلى :

```
BEGIN
   read(file name, variable name 1);
   read(file name, variable name 2);
       .
       .
       .
   read(file name, variable name n)
END;
```

مرة أخرى ، يجب أن يكون مفهوما أن نوع البيانات الذى يصاحب كل متغير يجب أن يكون من نفس نوع احتياطى الملف ، أو من نوع متوافق معه .

### مثـال (١١-١٦)

رأينا فى مثال (١١ - ١٣) برنامجا بلغة البسكال ، ينتج ملفا محتويا على تسلسل من الأعداد ومربعاتها وجذورها التربيعية . دعنا نعتبر الآن برنامجا يقرأ هذا الملف . مثل هذا البرنامج مدون أدناه :

```
PROGRAM readfile3(output,data);
(* THIS PROGRAM READS A DATA FILE CONTAINING THE FIRST
   100 INTEGERS, THEIR SQUARES AND THEIR SQUARE ROOTS *)
VAR data : FILE OF real;
    value,square,root : real;
BEGIN  (* main action statements *)
   reset(data);
   WHILE NOT eof(data) DO
      BEGIN
         read(data,value,square,root);
         writeln(value:6:2,square:8:2,root:6:2)
      END
END.
```

لاحظ أن هذا البرنامج أكثر عمومية عن مثيله الموجود فى مثال (١١ - ١٣) ، حيث إنه ليس مقيدا على ملف سبق تحديد حجمه . لاحظ أيضا مظهر المتغيرات المتعددة فى عبارة read .

## ٧ - تجديد الملف 7. UPDATING A FILE

يمكن إنتاج ملف تتابعى داخل برنامج البسكال كملف مدخلات فقط أو ملف مخرجات فقط ، ولايمكن إنتاج ملفى مدخلات ومخرجات فى نفس البرنامج . ( لاحظ على أية حال أنه يمكن معاملة ملف معين كملف مدخلات فى

برنامج ، وكملف مخرجات فى برنامج آخر ) . وعلى هذا ... فليس من الممكن قراءة مكونات ملف وتعديلها وكتابة المكونات المعدلة على نفس الملف . وبالمثل ليس من الممكن قراءة ملف حتى نهايته ، ثم إضافة سجلات إضافية على نفس الملف .

تميل هذه القيود إلى تعقيد عملية تجديد الملف . ويمكن تجديد الملف بطريقة مباشرة نسبية على أية حال ، وذلك باستخدام ملفين داخل البرنامج . أحد الملفين يكون الملف الأصلى المراد تجديده ( الملف القديم ) . وسوف يعامل هذا الملف كملف مدخلات . أما الملف الثانى ( الملف الجديد ) ، فإنه يحتوى على المعلومات المجددة ، وبعض هذه المعلومات ينسخ مباشرة من الملف القديم ، والبعض الآخر يتم إدخاله من وحدة مدخلات قياسية ( مثل نهاية طرفية للمشاركة الزمنية ) . وسوف يعامل على ذلك الملف الجديد على أنه ملف مخرجات .

ولتجديد ملف قديم ، فإننا نجرى الخطوات التالية لكل مكون من مكونات الملف .

١ – يقرأ مكون الملف داخل ذاكرة الكمبيوتر ، ويحدد ما إذا كان سيجدد أم لا . ويمكن أن يتحقق ذلك بمقارنة أحد الأدلة لعنصر بيانات ، مثل رقم العميل مع قيمة تم إدخالها من وحدة مدخلات قياسية . ( تحدد القيمة التى يتم إدخالها من وحدة المدخلات القياسية المكون الثانى الذى ستجرى عليه عملية التجديد ) .

أ – إذا كان مكون الملف الحالى لايحتاج إلى تجديد ، فيتم نسخه مباشرة فى الملف الجديد .

ب – إذا كان التجديد مطلوبا ، فيتم إدخال المعلومات الجديدة من وحدة المدخلات ودمجها مع المعلومات القديمة بالطريقة المناسبة ، ثم يكتب مكون الملف الجديد فى الملف الجديد . وبعد ذلك يتم إدخال قيمة أخرى من وحدة المدخلات تحدد مكون الملف الثانى المطلوب تجديده .

٢ – تتكرر هذه العملية حتى تقرأ كل مكونات الملف من الملف القديم ( أى حتى يتم الوصول إلى نهاية الملف ) .

٣ – إذا كان مطلوبا إضافة مكونات جديدة إلى الملف الأصلى ، فيجب إدخالها عند ذلك الوقت من وحدة المدخلات بالترتيب المناسب . ويكتب كل مكون من مكونات الملف الجديدة فى الملف الجديد كما يدخل من وحدة المدخلات تماما .

ولكى تعمل هذه الخطوات بطريقة مناسبة يجب إدخال المعلومات الجديدة متتالية من وحدة المدخلات ، وفى الترتيب الصحيح . ويجب أن يتوافق الترتيب مع الترتيب المستخدم فى تخزين مكونات الملف داخل الملف الأصلى . فمثلا إذا كانت مكونات الملف هى سجلات العملاء المخزنة فى ترتيب تصاعدى طبقا لرقم العميل ، فيجب أن تدخل المعلومات الجديدة ترتيبا تصاعديا طبقا لرقم العميل أيضا .

## مثـــال (١١–١٧)

*نسخ ملف* . فيما يلى برنامجا موجزا يتسبب فى نسخ مكونات ملف اسمه old file فى ملف آخر اسمه new file .

```
PROGRAM sample(newfile,oldfile);
(* THIS PROGRAM COPIES THE CONTENTS OF ONE FILE TO ANOTHER *)
TYPE account = RECORD
        custno : 1..9999;
        oldbalance : real;
        newbalance : real;
        payment : real;
     END;
```

(تكملة البرنامج فى الصفحة التالية)

```
VAR newfile,oldfile : FILE OF account;
    newaccount,oldaccount : account;
BEGIN   (* main action block *)
   reset(oldfile);
   rewrite(newfile);
   WHILE NOT eof(oldfile) DO
      BEGIN
         read(oldfile,oldaccount);
         newaccount := oldaccount;
         write(newfile,newaccount)
      END
END.
```

وليس من الضرورى فى واقع الأمر وجود متغيرين منفصلين من نوع السجل oldaccount , newaccount داخل تخطيط البرنامج . فأحد هذين المتغيرين يكون كافيا . وعلى هذا ... فإن المجموعة الإجرائية الرئيسية يمكن أن تكتب على النحو التالى :

```
BEGIN   (* main action block *)
   reset(oldfile);
   rewrite(newfile);
   WHILE NOT eof(oldfile) DO
      BEGIN
         read(oldfile,newaccount);
         write(newfile,newaccount)
      END
END
```

وعلى أية حال ... فإن استخدام متغيرين مختلفين يميز الطريقة التى تنقل بها المكونات الفردية للملف من ملف إلى آخر ، حيث إن محتويات oldaccount يمكن تعديلها بطريقة معينة قبل نقلها إلى newaccount .

## مثـال (١١-١٨)

دعنا نعتبر الآن عملية نقل إحدى المكونات ، أو العديد من المكونات ، إلى ملف موجود فعلا . ويكون ضروريا مرة أخرى قراءة كل المكونات من oldfile فى newfile . وبعد الانتهاء من نقل كل المكونات القديمة ، يتم إدخال المكونات الجديدة من لوحة المفاتيح . وتستمر هذه العملية حتى يتم إدخال القيمة 9999 كقيمة للمتغير custno .

وفيما يلى تخطيطا للبرنامج :

```
PROGRAM sample(input,output,newfile,oldfile);
(* TRANSFER THE CONTENTS OF ONE FILE TO ANOTHER,
   THEN APPEND ADDITIONAL COMPONENTS TO THE NEW FILE *)
TYPE account = RECORD
                  custno : 1..9999;
                  oldbalance : real;
                  newbalance : real;
                  payment : real;
               END;
```

(تكملة البرنامج فى الصفحة التالية)

```
VAR newfile,oldfile : FILE OF account;
    newaccount : account;
    custno : 1..9999;

PROCEDURE readinput(VAR newaccount : account);
(* enter input data for one record *)
BEGIN
   .
   .
   (* enter input data from the keyboard for each record *)
   .
   .
END;


BEGIN   (* main action block *)
   reset(oldfile);
   rewrite(newfile);

   WHILE NOT eof(oldfile) DO   (* transfer old records *)
      BEGIN
         read(oldfile,newaccount);
         write(newfile,newaccount)
      END;   (* transfer of old records *)

   write('Customer number: ');
   readln(custno);
   WHILE custno < 9999 DO   (* enter new records *)
      BEGIN
         readinput(newaccount);
         write(newfile,newaccount);
         write('Customer number: ');
         readln(custno)
      END   (* enter new records *)
END.
```

لاحظ أن كل مكون جديد من مكونات الملف ( كل سجل جديد ) يمرر من الإجراء readinput إلى الجزء الرئيسى للبرنامج كمؤشر من نوع المتغير .

## مثـال (١١-١٩)

فيما يلى هيكلا لبرنامج بسكال يقرأ مكونات ملف متتالية من ملف اسمه oldfile ، وتجديدها إذا كان هذا مطلوبا ، ثم نسخها فى الملف المسمى newfile . ويستمر تجديد المكونات المختلفة حتى تدخل قيمة 9999 كقيمة للمتغير custno . وأى مكونات تظل فى الملف القديم oldfile تنقل عند ذلك مباشرة إلى newfile دون أى فرصة لتجديدها .

لاحظ أن كل مكونات oldfile تنقل إلى newfile . بعضها ينقل بعد تجديده ، وبعضها ينقل كما هو دون أى تغـيير .

ولكى تعمل هذه الطريقة بطريقة مناسبة ، فمن الضرورى إدخال القيم المتتالية للمتغير custno فى ترتيب تصاعدى . وهذا يؤكد أن تسلسل التغيير متطابق مع الترتيب المناظرلتخزين مكونات الملف فى oldfile .

```
PROGRAM sample(input,output,newfile,oldfile);

(* THIS PROGRAM READS SUCCESSIVE FILE COMPONENTS FROM A
   DATA FILE, UPDATES EACH COMPONENT IF NECESSARY,
   AND THEN WRITES THE COMPONENTS TO A NEW DATA FILE *)

TYPE account = RECORD
                 custno : 1..9999;
                 oldbalance : real;
                 newbalance : real;
                 payment : real;
              END;
VAR newfile,oldfile : FILE OF account;
    newaccount,oldaccount : account;
    custno : 1..9999;

BEGIN   (* main action statements *)
   reset(oldfile);
   rewrite(newfile);
   write('Customer number: ');
   readln(custno);
   WHILE NOT eof(oldfile) DO
     BEGIN
        read(oldfile,oldaccount);
        newaccount := oldaccount;
        IF (custno < 9999) AND (oldaccount.custno = custno) THEN
           BEGIN
              .
              .
              (* accept appropriate input from the keyboard and
                 make changes in a file component as required *)
              .
              .
              write('Customer number: ');
              readln(custno)
           END;   (* IF custno *)
        write(newfile,newaccount)
     END   (* WHILE NOT eof *)
END.
```

فى بعض الأحيان يكون من الضرورى نقل محتويات الملف إلى إجراء كمؤشر . فى مثل هذه الحالات يجب أن يمرر الملف كمؤشر متغير variable دائما . وهذا يلغى الحاجة لإنتاج نسخة مطولة من الملف ، كما هو الحال إذا ماتم تمرير الملف كمؤشر قيمة . سوف نرى توضيحا لهذه النقطة فى المثال التالى :

## مثـال (١١-٢٠)

تجديد *نظام فواتير عملاء* . نعتبر الآن برنامجا كاملا بلغة البسكال ، يسمح لنا بتجديد ملف بيانات يستخدم فى حفظ نظام فواتير العملاء . يفترض أن مثل هذا التجديد يجرى كل فترة زمنية معينة ( كل شهر مثلا ) .

دعنا نفحص كل السجلات أثناء كل عملية تجديد . يسمح هذا الإجراء بالمدفوعات الحديثة أن تخصم من الحساب الخاص بالعملاء المدنيين . ويمكن عند ذلك تغيير حالة الحساب طبقا لهذه المدفوعات .

يحتوى التجديد على تسجيل أحدث المدفوعات وتاريخها ، وحساب الموازنة الجديدة ، وتحديد حالة حساب كل عميل .

افرض أن سجل كل عميل يحتوى على عناصر المعلومات التالية :

الاسم ، ورقم الحساب ( رقم العميل ) ، وحالة الحساب ( جار أو متأخر أو مقصر ) ، والموازنة السابقة ، والموازنة الجديدة ، ومادفع حاليا ، وتاريخ الدفع . وتوضيحات السجلات المناسبة هى كمايلى :

```
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..30] OF char;
     date = RECORD
               month : 1..12;
               day : 1..31;
               year : 1900..2100
            END;
     account = RECORD
                  name : line;
                  custno : 1..9999;
                  custtype : status;
                  oldbalance : real;
                  newbalance : real;
                  payment : real;
                  paydate : date
               END;
```

لاحظ أن توضيحات السجل هى مثل توضيحات المعطاه فى مثال (١٠ – ٢١) ، فيما عدا ان عنوان العميل حذف هنا للتبسيط .

وتعتبر حالة كل حساب من حسابات العملاء جارية ، إلا إذا لم تكن هناك موازنة . وفى هذه الحالة تعتمد حالة الحساب على حجم أحدث مبلغ تم دفعه ، وتطبق القواعد التالية فى ذلك .

١ – إذا كان المبلغ المدفوع حاليا أكبر من صفر ، وأقل من ١٠ ٪ من الموازنة السابقة ، فيعتبر الحساب متأخرًا .

٢ – إذا كانت هناك موازنة قائمة ، وكانت القيمة المدفوعة حاليا تساوى صفرا ، فيعتبر الحساب مقصراً

( وهذا هو نفس الشئ أيضا مثل المثال رقم (١٠ – ٢١) ) .

والعملية الكاملة لكل تجديد للملف تصبح على النحو التالى :

١ – تكتب كل السجلات الموجودة فى ملف البيانات القديم ( وذلك لتقديم نسخة من الملف الأصلى ) .

٢ – يستمر العمل متتاليا خلال الملف القديم ، مع تجديد كل السجلات التى بها موازنات قائمة .

٣ – يكتب كل سجل فى ملف البيانات الجديد . ( لاحظ أن هذا يشمل كل السجلات ، بغض النظر عما إذا حدث لها تجديد أم لا ) . وسوف تكتب السجلات التى أجرى لها تجديد فى صورتها الجديدة المعدلة ، أما السجلات الأخرى ، فسوف تنسخ بنفس الصورة التى كانت عليها فى الملف القديم

٤ – تكتب كل السجلات فى ملف البيانات الجديد .

ويبدو التكوين الهيكلى الشامل لهذا البرنامج على النحو التالى :

```
PROGRAM billing2(input,output,newfile,oldfile);
TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..30] OF char;
     date = RECORD
              month : 1..12;
              day : 1..31;
              year : 1900..2100
           END;
     account = RECORD
                 name : line;
                 custno : 1..9999;
                 custtype : status;
                 oldbalance : real;
                 newbalance : real;
                 payment : real;
                 paydate : date
              END;
     datafile = FILE OF account;
VAR newfile,oldfile : datafile;
    newaccount,oldaccount : account;

PROCEDURE writefile(VAR data : datafile);
VAR custaccount : account;
BEGIN
   .
   .
   (* write the contents of the file
      represented by the variable parameter data *)
   .
   .
END;

PROCEDURE update(VAR newaccount : account);
BEGIN
   .
   .
   (* accept appropriate input from the keyboard and
       make changes in a file component as required *)
   .
   .
END;

BEGIN   (* main action block *)
   writeln('CUSTOMER BILLING SYSTEM - FILE UPDATE');
   writeln;
   writeln('OLD DATA FILE');
   writeln;
   writefile(oldfile);

   BEGIN   (* update the old file *)
```

(تكملة البرنامج فى الصفحة التالية)

```
        writeln;
        writeln('DATA UPDATE');
        writeln;
        reset(oldfile);
        rewrite(newfile);
        WHILE NOT eof(oldfile) DO
           BEGIN
              read(oldfile,oldaccount);
              newaccount := oldaccount;
              IF oldaccount.newbalance > 0 THEN update(newaccount);
              write(newfile,newaccount)
           END   (* WHILE NOT eof(oldfile) *)
     END;   (* file update *)

     writeln;
     writeln('NEW DATA FILE');
     writeln;
     writefile(newfile)
  END.
```

يستخدم هذا البرنامج إجرائين ، هما : update , writefile . يقرأ أول هذين الإجرائين write file السجلات الموجودة فى ملف بيانات ، ثم يكتبها فى وحدة مخرجات ( مثل الطابع ) . حيث إن هذا الإجراء يستخدم مع كل من ملفى البيانات ، فإننا نستخدم مؤشراً متغيراً يسمى data داخل توضيح الإجراء . لاحظ أن data يمثل ملفا من نفس نوع الملف data file كما هو معرف فى المجموعة الرئيسية للبرنامج . ويظهر الإجراء الفعلى writefile على النحو التالى :

```
PROCEDURE writefile(VAR data : datafile);

(* THIS PROCEDURE CAUSES THE CONTENTS OF A DATA FILE TO BE WRITTEN OUT *)

VAR custaccount : account;
BEGIN
   reset(data);
   WHILE NOT eof(data) DO
      BEGIN
         read(data,custaccount);
         WITH custaccount DO
            BEGIN
               writeln;
               write('Name:    ',name);
               writeln('    Customer Number: ',custno:4);
               writeln;
               write('Old balance: ',oldbalance:7:2);
               write('   Current payment: ',payment:7:2);
               writeln('   New balance: ',newbalance:7:2);
               write('Payment date: ',paydate.month:2,'/');
               writeln(paydate.day:2,'/',paydate.year:4);
               writeln;
               write('Account status: ');
               CASE custtype OF
                  current    : writeln('CURRENT');
                  overdue    : writeln('OVERDUE');
                  delinquent : writeln('DELINQUENT')
               END;
               writeln
            END   (* WITH custaccount *)
      END   (* WHILE NOT eof(data) *)
END;
```

المنطق مباشر ، وليس فى حاجة إلى مناقشة .

اعتبر الآن الإجراء الثانى update . الغرض من هذا الإجراء هو إدخال أحدث مبلغ مدفوع لعميل معين ، وتضبيط موازنة العميل ، وبعد ذلك تحديد الحالة المناسبة لحساب العميل طبقا للموازنة السابقة ، وحجم المبلغ المدفوع حاليا .

ويمكن كتابة إجراء update على النحو التالى :

```
PROCEDURE update(VAR newaccount : account);

(* THIS PROCEDURE ENTERS NEW DATA FROM THE KEYBOARD
          AND THEN UPDATES EACH RECORD AS NECESSARY *)

VAR slash : char;
BEGIN
   WITH newaccount DO
      BEGIN
         oldbalance := newbalance;
         writeln('Customer number: ',custno:4);
         write('Current balance: ',oldbalance:7:2);
         write('   Current payment: ');
         readln(payment);
         IF payment > 0 THEN
            BEGIN
               write('Payment date: ');
               WITH paydate DO
                  readln(month,slash,day,slash,year);
            END;
         writeln;
         newbalance := oldbalance - payment;
         IF payment >= 0.1*oldbalance
            THEN custtype := current
            ELSE IF payment > 0 THEN custtype := overdue
                                ELSE custtype := delinquent
      END  (* WITH newaccount *)
END;
```

إذا جمعنا كل الأجزاء الآن مع بعضها ، فيظهر البرنامج الكامل على النحو التالى :

```
PROGRAM billing2(input,output,newfile,oldfile);

(* THIS PROGRAM UPDATES A DATA FILE BY READING AN OLD FILE,
   UPDATING EACH RECORD, AND THEN WRITING THE UPDATED
   RECORDS ONTO A NEW FILE *)

TYPE status = (current,overdue,delinquent);
     line = PACKED ARRAY [1..30] OF char;
     date = RECORD
               month : 1..12;
               day : 1..31;
               year : 1900..2100
            END;
```

(تكملة البرنامج فى الصفحة التالية)

```
     account = RECORD
                 name : line;
                 custno : 1..9999;
                 custtype : status;
                 oldbalance : real;
                 newbalance : real;
                 payment : real;
                 paydate : date
              END;
     datafile = FILE OF account;
VAR newfile,oldfile : datafile;
    newaccount,oldaccount : account;

PROCEDURE writefile(VAR data : datafile);

(* THIS PROCEDURE CAUSES THE CONTENTS OF A DATA FILE TO BE WRITTEN OUT *)

VAR custaccount : account;
BEGIN
   reset(data);
   WHILE NOT eof(data) DO
      BEGIN
         read(data,custaccount);
         WITH custaccount DO
            BEGIN
               writeln;
               write('Name:   ',name);
               writeln('   Customer Number: ',custno:4);
               writeln;
               write('Old balance: ',oldbalance:7:2);
               write('   Current payment: ',payment:7:2);
               writeln('   New balance: ',newbalance:7:2);
               write('Payment date: ',paydate.month:2,'/');
               writeln(paydate.day:2,'/',paydate.year:4);
               writeln;
               write('Account status: ');
               CASE custtype OF
                  current    : writeln('CURRENT');
                  overdue    : writeln('OVERDUE');
                  delinquent : writeln('DELINQUENT')
               END;
               writeln
            END  (* WITH custaccount *)
      END  (* WHILE NOT eof(data) *)
END;  (* writefile *)

PROCEDURE update(VAR newaccount : account);

(* THIS PROCEDURE ENTERS NEW DATA FROM THE KEYBOARD
          AND THEN UPDATES EACH RECORD AS NECESSARY *)

VAR slash : char;
BEGIN
   WITH newaccount DO
      BEGIN
         oldbalance := newbalance;
         writeln('Customer number: ',custno:4);
         write('Current balance: ',oldbalance:7:2);
         write('   Current payment: ');
         readln(payment);
```

(تكملة البرنامج فى الصفحة التالية)

```
          IF payment > 0 THEN
             BEGIN
                write('Payment date: ');
                WITH paydate DO
                   readln(month,slash,day,slash,year);
             END;
          writeln;
          newbalance := oldbalance - payment;
          IF payment >= 0.1*oldbalance
             THEN custtype := current
             ELSE IF payment > 0 THEN custtype := overdue
                                 ELSE custtype := delinquent
       END   (* WITH newaccount *)
END;   (* update *)

BEGIN   (* main action block *)
   writeln('CUSTOMER BILLING SYSTEM - FILE UPDATE');
   writeln;
   writeln('OLD DATA FILE');
   writeln;
   writefile(oldfile);

   BEGIN   (* update the old file *)
      writeln;
      writeln('DATA UPDATE');
      writeln;
      reset(oldfile);
      rewrite(newfile);
      WHILE NOT eof(oldfile) DO
         BEGIN
            read(oldfile,oldaccount);
            newaccount := oldaccount;
            IF oldaccount.newbalance > 0 THEN update(newaccount);
            write(newfile,newaccount)
         END   (* WHILE NOT eof(oldfile) *)
   END;   (* file update *)

   writeln;
   writeln('NEW DATA FILE');
   writeln;
   writefile(newfile)
END.
```

أفترض الآن أن البرنامج يستخدم لتشغيل ملف بيانات بسيط يحتوى على ستة حسابات . يبدأ هذا البرنامج بطباعة محتويات ملف البيانات الأصلى oldfile كما يلى :

```
CUSTOMER BILLING SYSTEM - FILE UPDATE

OLD DATA FILE


Name:  Alan B. Adams                 Customer Number:    1

Old balance:  128.32   Current Payment:  128.32   New balance:     0.00
Payment date: 11/27/1984

Account Status: CURRENT
```

(تكملة البرنامج فى الصفحة التالية)

```
Name:   James Brown                    Customer Number:   2

Old balance:  189.00   Current Payment:  100.00  New balance:    89.00
Payment date: 11/18/1984

Account Status: CURRENT

Name:   Janet P. Davis                 Customer Number:   3

Old balance:  212.18   Current Payment:    0.00  New balance:  212.18
Payment date:  9/ 4/1984

Account Status: DELINQUENT

Name:   William D. McDonald            Customer Number:   4

Old balance:  200.00   Current Payment:   65.00  New balance:  135.00
Payment date: 11/10/1984

Account Status: CURRENT

Name:   Phyllis W. Smith               Customer Number:   5

Old balance:  289.42   Current Payment:   20.00  New balance:  269.42
Payment date: 11/21/1984

Account Status: OVERDUE

Name:   Richard L. Warren              Customer Number:   6

Old balance:   70.00   Current Payment:   50.00  New balance:    20.00
Payment date: 11/15/1984

Account Status: CURRENT
```

بعد كتابة محتويات ملف البيانات القديم ، يلقن البرنامج المستفيد ليدخل بيانات جديدة لكل عميل لاتكون موازناتها القائمة مساوية للصفر . وعلى هذا ... يجدد خمسة من السجلات . والحوار المتداخل لتجديد البيانات يبدو فى الصورة التالية ( لاحظ أن كل استجابات المستفيد موضوع تحتها خط ) .

```
DATA UPDATE

Customer number:   2
Current balance:   89.00   Current payment: 50.00
Payment date: 12/12/1984

Customer number:   3
Current balance:   212.18  Current payment: 212.18
Payment date: 12/3/1984

Customer number   4
Current balance:   135.00  Current payment: 10.00
Payment date: 12/18/1984

Customer number:   5
Current balance:   269.42  Current payment: 0

Customer number:   6
Current balance:   20.00   Current payment: 20.00
Payment date: 12/7/1984
```

لاحظ أن ملقنات البرنامج لمعلومات جديدة فقط للعملاء الذين تكون موازناتهم الحالية أكبر من الصفر . لاحظ أن البرنامج سوف يلقن أيضا لإدخال تاريخ المبلغ المدفوع إذا كان هناك مبلغ مدفوع أكبر من الصفر فقط .

وبعد إدخال البيانات الجديدة وإتمام التجديد ، يكتب ملف العملاء الجديد . وتظهر محتويات هذا الملف على النحو التالى :

```
NEW DATA FILE

Name:   Alan B. Adams                    Customer Number:    1

Old balance:  128.32   Current Payment:  128.32   New balance:    0.00
Payment date: 11/27/1984

Account Status: CURRENT

Name:   James Brown                      Customer Number:    2

Old balance:   89.00   Current Payment:   50.00   New balance:   39.00
Payment date: 12/12/1984

Account Status: CURRENT

Name:   Janet P. Davis                   Customer Number:    3

Old balance:  212.18   Current Payment:  212.18   New balance:    0.00
Payment date:  9/ 4/1984

Account Status: CURRENT
```

(تكملة البرنامج فى الصفحة التالية)

```
Name:   William D. McDonald           Customer Number:    4

Old balance:  135.00  Current Payment:   10.00   New balance:  125.00
Payment date: 12/18/1984

Account Status: OVERDUE


Name:   Phyllis W. Smith              Customer Number:    5

Old balance:  269.42  Current Payment:    0.00   New balance:  269.42
Payment date: 11/21/1984

Account Status: DELINQUENT


Name:   Richard L. Warren             Customer Number:    6

Old balance:   20.00  Current Payment:   20.00   New balance:    0.00
Payment date: 12/ 7/1984

Account Status: CURRENT
```

لاحظ أن أول سجل يظل كماهو بدون أى تغيير ، حيث إن أول عميل ( Alan B. Adams ) ليس لديه موازنة قائمة أثناء فترة التجديد هذه .

وبعد إتمام تنفيذ البرنامج ، يجب أن يحذف oldfile ، ويعاد تسمية newfile باسم oldfile . ويسمح ذلك بتجديد الملف الذى تم تجديده خلال هذه الدورة فى المستقبل ( أو فى الدورة التالية ) .

## ٨ – ملفات النصوص 8. TEXT FILES

يمكن استخدام نوع آخر من أنواع الملفات فى البسكال ، وهو ملفات النصوص text files . وهذا الملف من النوع الحرفى ، مع وجود محدد لنهاية السطر فى أماكن مختلفة لتمييز كل سطر من أسطر النص عن بقية سطور النص . ( وعادة مايحتوى محدد نهاية السطر على تغذية للسطر ، تتبع عودة العربة ) . وعلى هذا ... يتكون ملف النص من أسطر متعددة لبيانات من النوع الحرفى .

وقد يحدث تجميع للرموز الموجودة فى أى سطر ( أى تفصل عن بعضها بواسطة فراغات ) . ويمكن تفسير مجموعة الرموز الفردية بإحدى طرق عديدة . فمثلا مجموعة من الأرقام الصحيحة يمكن تفسيرها ككمية صحيحة أو حقيقية . وتتحدد طريقة تفسير هذه المجموعات لعبارة read أو عبارة write المصاحبة ( المزيد عن ذلك يذكر فيما بعد ) .

ويمكن تعريف ملف النص بسهولة ، وذلك بإقراره كمعرف ( أى كمتغير يمثل الملف ) من نوع النص . ويمكن أن يعبر رسميا عن التوضيح كما يلى :

VAR *file name* : text

حيث file name هو معرف يمثل اسم ملف النص . ويمكن بالطبع وجود عدة أسماء ملفات فى توضيح واحد .

## مثـال (١١-٢١)

افرض أنه يجب على برنامج البسكال أن يستخدم ملفى النصوص newtext , oldtext . يظهر تعريف الملفين كما يلى :

```
VAR oldtext,newtext : text;
```

لاحظ أنه ليس من الضرورى كتابة "FILE OF text" داخل التوضيح . وكل ملف نص له احتياطى ملف مصاحب له ، مثل أى ملف آخر . وحيث إن ملف النص من النوع الحرفى ، فإن احتياطى ملف النص يمثل رمزا واحدا . ويمكن استخدام الإجرا ءين القياسيين rewrite , reset مع ملفات النصوص ، كما يستخدمان تماما مع الأنواع الأخرى من الملفات . وتستخدم عبارة reset لإعداد الملف للقراءة . وعلى هذا ... فإن العبارة التالية :

```
reset(file name);
```

تتسبب فى تحديد أول رمز لاحتياطى الملف ، على ألا يكون الملف فارغا . والدالة القياسية :

```
eof(file name)
```

تضع قيمة مناسبة عند إعداد الملف .

وبالمثل تعد عبارة rewrite للكتابة فى الملف . وعلى هذا ... فإن العبارة :

```
rewrite(file name);
```

تمحى النص الموجود فعلا فى ملف النص ( إذا ماكان موجودا ملف ) ، ثم تحدد بداية ملف النص .

كما يمكن استخدام الإجرائين القياسيين أيضا مع ملفات النصوص ، مثل استخدامهما مع أنواع الملفات الأخرى تماما . وعلى هذا ... فالعبارة

```
get(file name);
```

تتسبب فى تحديد الرمز التالى فى ملف النص لاحتياطى الملف . ( فى العادة يكون قد تم تشغيل الرمز السابق قبل قراءة الرمز التالى ) .

```
put(file name);
```

أيضا فى كتابة الرمز الموجود حاليا فى احتياطى الملف فى الملف . وكقاعدة عامة ... يسبق عبارة put عبارة تحديد ، وذلك لوضع رمز جديد فى احتياطى الملف . وسوف يكتب هذا الرمز الجديد فى ملف النص كأثر لعبارة put .

وتستخدم معظم البرامج التى تستخدم ملفات نصوص الإجراءات القياسية writeln , write , readln , read ، بدلا من put , get . دعنا على ذلك نعيد فحص هذه الإجراءات فى ضوء مناقشتنا الحالية لملفات النصوص .

إذا ماإستخدمت العبارة التالية :

```
read(file name, variable name);
```

مع ملف نص ، فيعتمد الإجراء المأخوذ على نوع بيانات المتغير .

إذا كان المتغير من النوع الحرفى ، فيتم تشغيل عنصر البيانات بالطريقة المعتادة ، أى يتحدد الرمز الموجود فى احتياطى الملف للمتغير ، يتبعه قراءة رمز جديد من ملف النص ، وتحديده لاحتياطى الملف . أما إذا كان المتغير من النوع الصحيح أو النوع الحقيقى ، فيقرأ عدداً كافياً متتالياً من الرموز ليسمح بتكوين كمية صحيحة أو حقيقية . عند ذلك يتم تحديد هذه الكمية للمتغير ، ثم يتم تحديد الرمز التالى فى الملف لاحتياطى الملف .

ويمكن أن تظهر أسماء متغيرات متعددة فى عبارة read واحدة . وتفسر مثل هذه العبارة كتسلسل من عبارات read ، كل منها يحتوى على اسم متغير واحد . وعلى هذا ... فالعبارة :

```
read(file name, variable name 1, variable name 2, . . . , variable name n);
```

تفسر كما يلى :

```
BEGIN
   read(file name, variable name 1);
   read(file name, variable name 2);
      .
      .
      .
   read(file name, variable name n)
END;
```

إذا مامثلت المتغيرات كميات عددية متتالية ، فعلى ذلك يعتمد فصل كميه عددية عن كمية أخرى على اكتشاف الفراغات التى تسبقها ، أو اكتشاف رموز سطر التحكم كما سبق ذكره فى الفصل الرابع ( انظر القسم ٢ من الفصل الرابع ) .

وتشبه عبارة readln عبارة read . فى هذه الحالة تتسبب عبارة readln واحدة فى استمرار القراءة حتى يظهر المحدد الخاص بنهاية السطر ( eoln ) . وعلى هذا ... فإن أى read تالية أو readln تالية تبدأ فى السطر التالى ( انظر القسم ٣ من الفصل الرابع ) .

ولنكون محددين أكثر ، فإن العبارة :

```
readln(file name);
```

تكافئ :

```
BEGIN
   WHILE NOT eoln(file name) DO get(file name);
   get (file name)
END;
```

( لاحظ أن اخر get تتسبب فى تحديد أول رمز فى السطر التالى لاحتياطى الملف ) . وأكثر من هذا ... العبارة :

```
readln(file name, variable name 1, variable name 2, . . . , variable name n);
```

تفسر كما يلى :

```
BEGIN
   read(file name, variable name 1, variable name 2, . . . , variable name n);
   readln(file name)
END;
```

والعبارة القياسية :

```
eoln(file name)
```

هى دالة بوليان قياسية تعود بقيمة خاطئ false ، إلا إذا كان الرمز الحالى فى الملف هو مؤشر نهاية الملف . وعلى هذا ... تظل الدالة eoln خاطئة false حتى تكتشف نهاية الملف ، والتى تسبب فى جعلها صحيحة true . وبعد ذلك تصبح الدالة خاطئة false مرة أخرى عندما يفحص رمز آخر بعد اكتشاف نهاية السطر ( انظر قسم ٤ من الفصل الرابع ، وقسم ١ من الفصل التاسع ) .

دعنا الآن نوجه انتباهنا إلى عبارات write and writeln

إذا استخدمت العبارة .

```
write(file name, output item)
```

مع ملف ، فيعتمد الإجراء المتخذ على طبيعة عنصر المخرجات . إذا كان عنصر المخرجات من النوع الحرفى ( أى أنه ثابت أو متغير من النوع الحرفى ) ، فعند ذلك يتحدد الرمز لاحتياطى الملف ، ويكتب فى الملف . أما إذا مامثل عنصر المخرجات نوع بيانات آخر ( أى متغير بوليان ، أو ثابت عددى ، أو متغير عددى ، أو تعبير عددى ) فعند ذلك تتحول قيمة عنصر المخرجات إلى الرموز التى تحتويها . ونكتب هذه الرموز بعد ذلك تلقائيا فى ملف النص .

ويمكن أن تظهر عناصر مخرجات متعددة فى عبارة write واحدة . وتفسر مثل هذه العبارة على أنها تسلسل من عبارات write ، كل منها يحتوى على عنصر مخرجات واحد . وعلى هذا ... فإن العبارة .

```
write(file name, output item 1, output item 2, . . . , output item n);
```

تفسر على أنها :

```
BEGIN
   write(file name, output item 1);
   write(file name, output item 2);
       .
       .
       .
   write(file name, output item n)
END;
```

انظر القسم ( ٥ من الفصل الرابع )

تتسبب العبارة .

```
writeln(file name);
```

فى كتابة تحديد لنهاية سطر فى الملف . وعلى هذا ... فأى مخرجات تالية لملف النص تبدأ على السطر التالى . وأكثر من ذلك ... العبارة

```
writeln(text file, output item 1, output item 2, . . . , output item n);
```

تفسر كما يلى :

```
BEGIN
   write(file name, output item 1, output item 2, . . . , output item n);
   writeln(file name)
END;
```

وتسمح بذلك لعناصر مخرجات متعددة أن توضع على نفس السطر ، يتبعها تحديد لنهاية السطر ( انظر القسم ٦ من الفصل الرابع ) .

وأخيرا يمكن استخدام الإجراء القياسى page مع أى ملف من ملفات النصوص ، وذلك بوضع أسـم الملف كمؤشـر أى :

```
page(file name);
```

ويتسبب استخدام هذا الإجراء فى بدء كل مخرج جديد فى بداية صفحة جديدة ، على أن يكون هذا المخرج موجها لوحدة يمكنها تمييز بداية الصفحة ( أى طابع أسطر ) ، وإلا فسوف تهمل عبارة page ( انظر القسم ٧ من الفصل الرابع ) .

## مثـال (١١-٢٢)

*نقل النصوص بين ملفات النصوص* . فيما يلى برنامج بسكال بسيطًا ينقل محتويات oldtext إلى newtext سطرا سطرا

( لاحظ أن كلا من oldtext و newtext هو ملف نص ) .

```
PROGRAM filetransfer(oldtext,newtext);

(* THIS PROGRAM TRANSFERS TEXT FROM ONE TEXT FILE
            TO ANOTHER ON A LINE-BY-LINE BASIS *)

VAR oldtext,newtext : text;
    x : char;

BEGIN
   reset(oldtext);
   rewrite(newtext);
   page(newtext);
   WHILE NOT eof(oldtext) DO
      BEGIN
         WHILE NOT eoln(oldtext) DO
            BEGIN   (* transfer one line of text *)
               read(oldtext,x);
               write(newtext,x)
            END;
         readln(oldtext);
         writeln(newtext)
      END
END.
```

تتسبب دورة WHILE - DO الداخلية فى قراءة النص الموجود فى سطر من أسطر oldtext، وكتابة فى newtext . ويحدث النقل رمزا رمزا ، وذلك بالنسبة لأى سطر من أسطر النص . يلى دورة WHILE - DO عبارة read التى تعد البرنامج لقراءة السطر الثانى من oldtext ، ثم عبارة writeln التى تضع محدد نهاية السطر فى السطر الحالى فى newtext .

وتتسبب دورة WHILE - DO الخارجية فى تكرار العمل سطرا سطرا ، حتى يكتشف محدد نهاية الملف فى oldtext .

ويوجد ملفان سبق توضيحهما فى البسكال ، هما : input , output ، ويستخدمان فى نقل بيانات مدخلات ومخرجات من وإلى وحدات مدخلات ومخرجات قياسية . وقد سبق لنا بالطبع استخدام هذه الملفات منذ ظهورهما فى الفصل الرابع من الكتاب ، بالرغم من أننا لم نعرفهما بأنهما ملفا نصوص حتى هذه النقطة .

وعند استخدام إجراءات أو دوال قياسية سبق ذكرها مع أحد هذين الملفين الذى سبق توضيحهما ، فلايكون من الضرورى أن يظهر اسم الملف كمؤشر . فإذا لم يتم تحديد المؤشر ، فيفترض input أو output طبقا للإجراء الخاص أو الدالة الخاصة . وعلى هذا ... فالإجراءات والدوال التالية يفترض أنها مطبقة على ملف نص output إذا لم يوجد مؤشر ملف صريح : write , writeln , page . أكثر من ذلك ... تعد input , output تلقائيا ، بحيث إن الإجرائين القياسيين reset , rewrite لايكون هناك حاجة لهما مع هذه الملفات .

## مثـال (١١-٢٣)

*إدخال وحفظ ملف نص* . فيما يلى برنامجًا يسمح بإدخال العديد من أسطر أحد النصوص من وحدة مدخلات ، وتخزينها فى ملف نص اسمه newtext . ( لاحظ أن هذا البرنامج صيغة أخرى من البرنامج الموجود فى مثال ١١ – ٢٢ ) .

```
PROGRAM entertext(input,newtext);
(* THIS PROGRAM ACCEPTS TEXT FROM AN INPUT DEVICE AND
     STORES IT IN A TEXT FILE ON A LINE-BY-LINE BASIS *)
VAR newtext : text;
    x : char;
BEGIN
   rewrite(newtext);
   WHILE NOT eof DO
      BEGIN
         WHILE NOT eoln DO
            BEGIN   (* enter and store one line of text *)
               read(x);
               write(newtext,x)
            END;
         readln;
         writeln(newtext)
      END
END.
```

عند تنفيذ هذا البرنامج ، يمكن إدخال عدد غير محدد من الأسطر من وحدة المدخلات ، وتخزينها فى newtext . ولفصل التنفيذ يجب إدخال محدد لنهاية الملف من وحدة المدخلات . وتختلف طبيعة هذا المحدد من منشأه لمنشأه أخرى ، بالرغم من أن رموز التحكم يتكرر استخدامها فى هذا الغرض .

## مثـال (١١-٢٤)

تشفير وإعادة تشفير النص . نعتبر الآن تطبيقا أكثر أهمية ، يشمل استخدام ملفات نصوص . دعنا نطور برنامجا متداخلا بلغة البسكال ، يعمل شفرة لعدة أسطر من أسطر أحد النصوص ، ثم يفك هذه الشفرة . ويحتوى البرنامج على قائمة تسمح باتخاذ أى إجراء من الإجراءات التالية :

١ – إدخال النص بواسطة لوحة المفاتيح ، وعمل شفرة للنص ، وتخزين النص وهو فى حالة هذه فى ملف نـص .

٢ – استرجاع الملف المسجل وهو فى حالة الشفرة ، وعرضه فى صورته هذه .

٣ – استرجاع الملف المسجل وهو فى حالة الشفرة ، وفك الشفرة ، ثم عرضه بعد فك الشفرة .

٤ – إنهاء الحسابات .

يمكن أن يحتوى جسم النص على العديد من الأسطر . وسوف تحفظ هذه الأسطر عندما يتم عمل شفرة للنص ، ثم يخزن ، وبعد ذلك تفك الشفرة ويعرض النص . وعلى أية حال ... فيحدث عمل الشفرة وفكها رمزا رمزا .

ولكى يتم عمل شفرة وفكها للنص ، يجب أن يدخل المستفيد رمزا فرديا "Key" ( سوف يلقن البرنامج لعمل ذلك إذا ما أختير عنصر القائمة 1 أو 3 ) . ويتم عمل شفرة لكل رمز وذلك بإضافة شفرته العددية للشفرة العددية للمفتاح key ثم تحديد الرمز الذى يمثله هذا المجموع ، أى :

```
z := chr(ord(x) + ord(key))
```

حيث z تمثل شفرة الرمز المكافئ للرمز الأصلى x .

وعند إضافة الشفرتين العدديتين يجب أن نكون حريصين ، حتى لايتعدى مجموعهما ١٢٧ ، حيث إن ١٢٧ هى أكبر قيمة لشفرة رمز قياسى ( بفرض مجموعة رموز ASCII ) . وعلى هذا ... إذا ماتعدت قيمة z العدد ١٢٧ ، فيجب أن تضبط قيمتها بطرح ١٢٧ من المجموع . وعلى هذا ... يمكننا كتابة :

```
y := ord(x) + ord(key);
IF y > 127 THEN y := y - 127;
z := chr(y);
```

تعمل هذه الطريقة جيدا وهى سهلة التنفيذ ، ويجب أن يكون مفهوما على أية حال أن بعض قيم z تنتج عنها رموز غير قابلة للطباعة ( أى تغذية أسطر ، أو عودة العربة ، أو beeps .. الخ ) . وسوف توضع هذه الرموز بالطبع فى النص المكتوب بالشفرة . وعلى هذا ... فيمكن أن يظهر النص المكتوب بالشفرة فى شكل غريب عند عرضه .

ويمكن إعادة فك الشفرة ، وذلك بعكس العملية السابقة ، أى :

```
y := ord(z) - ord(key);
IF y < 0 THEN y := y + 127;
x := chr(y);
```

وعند فك شفرة النص يكون من الضرورى بالطبع أن نستخدم نفس قيمة المفتاح key التى سبق استخدامها عند عمل الشفرة للنص الأصلى ، وإلا فإن فك الشفرة سوف يؤدى إلى نص غريب .

وفيما يلى تخطيطا هيكليا لمحتويات برنامج يوضح طريقة الحسابات الشاملة .

```
PROGRAM encode(input,output,code);
TYPE features = 1..4;
VAR code : text;
x,z,key : char;
y : -127..254;
choice : features;
PROCEDURE menu(VAR choice : features; VAR key : char);
BEGIN
   .
   .
   (* generate a menu, return the user's choice
      and a value for the key, if appropriate *)
   .
   .
END;
BEGIN   (* main action statements *)
   choice := 1;
   WHILE choice <> 4 DO
      menu(choice,key);
      CASE choice OF
       1 : BEGIN
                .
                .
                (* enter text, encode and store *)
                .
                .
             END;
      2,3 : BEGIN
                .
                .
                (* retrieve encoded text, decode
                   (if choice = 2) and display *)
                .
                .
             END;
        4 : ;
       END   (* CASE *)
    END   (* WHILE loop *)
END.
```

والطريقة الشاملة مباشرة ، وتعتمد على استخدام مكون CASE فى تنفيذ الاختيار المطلوب من القائمة . وبعد إتمام كل نشاط أساسى ( بعد كل ممر خلال مكون CASE ) ، يعود البرنامج إلى القائمة لأداء اختيار آخر ، حتى يتم اختيار البديل رقم ٤ .

وفيما يلى برنامج البسكال كاملا :

```
PROGRAM encode(input,output,code);

(* THIS IS A MENU-DRIVEN PROGRAM THAT ACCEPTS TEXT FROM AN INPUT DEVICE,
   ENCODES IT, STORES THE ENCODED TEXT IN A TEXT FILE, AND DISPLAYS THE
   TEXT, EITHER ENCODED OR DECODED *)

TYPE features = 1..4;

VAR code : text;
    x,z,key : char;
    y : -127..254;
    choice : features;

PROCEDURE menu (VAR choice : features; VAR key : char);

(* This procedure generates a menu and returns the user's choice *)

BEGIN
   writeln;
   writeln('E N C O D I N G / D E C O D I N G   T E X T');
   writeln;
   writeln('Program Features:');
   writeln;
   writeln('  1 - Enter text, encode and store');
   writeln;
   writeln('  2 - Retrieve encoded text and display');
   writeln;
   writeln('  3 - Retrieve encoded text, decode and display');
   writeln;
   writeln('  4 - End computation');
   writeln;
   write('Please enter your selection (1, 2, 3 or 4) -> ');
   readln(choice);
   writeln;
   IF (choice = 1) OR (choice = 3) THEN
      BEGIN
         write('Please enter the key (one character) -> ');
         readln(key)·
         writeln
      END
END;  (* menu *)

BEGIN  (* main action statements *)
   choice := 1;
   WHILE choice <> 4 DO
      BEGIN
         menu(choice,key);
         CASE choice OF
```

(تكملة البرنامج فى الصفحة التالية)

```
1 : BEGIN   (* enter text, encode and store *)
       rewrite(code);
       writeln('Enter text:');
       writeln;
       WHILE NOT eof DO
          BEGIN
             WHILE NOT eoln DO
                BEGIN
                   read(x);
                   y := ord(x) + ord(key);
                   IF y > 127 THEN y := y - 127
                   z := chr(y);
                   write(code,z)
                END;
             readln;
             writeln(code)
          END;  (* NOT eof *)
        reset(input)   (* eof=false *)
     END;

2,3 : BEGIN   (* retrieve encoded text, decode and display *)
         reset(code);
         WHILE NOT eof(code) DO
            BEGIN
               WHILE NOT eoln(code) DO
                  BEGIN
                     read(code,z);
                     IF choice = 2
                        THEN x := z
                        ELSE BEGIN
                                y := ord(z) - ord(key);
                                IF y < 0 THEN y := y + 127;
                                x := chr(y)
                             END;
                     write(x)
                  END;   (* NOT eoln *)
               readln(code);
               writeln
            END   (* NOT eof *)
      END;

   4 : ;

   END   (* CASE *)
 END;  (* WHILE choice <> 4 *)
writeln;
writeln('That''s all, folks!')
END.
```

يقدم الإجراء menu عبارات المدخلات والمدخلات اللازمة لإنتاج القائمة ، وإذا ما اختار المستفيد الاختيار ١ أو ٢ ، يلقن لاختيار مفتاح key رمزا واحدا . لاحظ أن هذا الإجراء يستخدم مؤشرين متغيرين هما : choice ، وهو من نوع key و features وهو من النوع الحرفى .

وتحتوى المجموعة الأساسية أساسا على مكون CASE به أربعة اختيارات مختلفة . أول اختيار يناظر Choice = 1 ، ويقدم التعليمات اللازمة لإدخال أسطر متعددة من البيانات وعمل الشفرة ، ثم التخزين سطرا سطرا . لاحظ أن الرموز التى أجرى لها الشفرة كتبت فى ملف النص Code .

ويستمر إدخال البيانات حتى يتم إدخال مايفيد انتهاء الملف من لوحة المفاتيح . ( وطريقة تحقيق ذلك تتغير من جهاز كمبيوتر لجهاز آخر ، بالرغم من أنه عادة مايستخدم رمز تحكم فى هذا الغرض ) . وبعد إمام إدخال البيانات ، يعاد إعداد reset ملف النص input ، بحيث إن شرط eof يكون خاطئا false مرة أخرى .

فإذا كانت قيمة Choice هى ٢ أو ٣ ، فإن الملف Code يعاد إعداده reset ، ويقرأ النص المشفر رمزا رمزا . إذا كان Code = 2 يكتب النص المشفر مباشرة فى وحدة المخرجات .

وإلا ( إذا كان Code = 3 ) تفك شفرة الرموز ، ثم تكتب فى وحدة المخرجات .

وعند تنفيذ البرنامج تنتج القائمة التالية :

```
E N C O D I N G / D E C O D I N G   T E X T

Program Features:

  1 - Enter text, encode and store

  2 - Retrieve encoded text and display

  3 - Retrieve encoded text, decode and display

  4 - End computation

Please enter your selection (1, 2, 3 or 4) ->
```

إذا ماأدخلت القيمة ١ أو ٣ كاختيار ، فينتج الملقن التالى للمفتاح key .

```
Please enter the key (one character) ->
```

افرض الآن أن المستفيد اختار إدخال عدة أسطر فى الملف . يجب على المستفيد أن يدخل القيمة ١ أولا للاختيار . افرض أن المستفيد أدخل الرمز C للمفتاح key ، وتبعه النص .

```
All digital computers, regardless of their size, are
basically electronic devices that can transmit, store and
manipulate information (i.e., data).
^Z
```

( الرموز Z^ فى نهاية النص تحدد نهاية الملف ) . يتم عمل شفرة لهذه المعلومات رمزا رمزا ، ثم تخزن فى ملف النص Code . ويتبع ذلك إعادة ظهور القائمة .

إذا مااختار المستفيد الآن الاختيار الثانى ، فتقرأ البيانات المشفرة من ملف النص Code ، وتعرض :

```
00c(-+-8%0c'31498)67oc6)+%6(0)77c3*c8,)-6c7->)oc%6)
&%7-'%00=c)0)'8632-'c():-')7c8,%8c'%2c86%271-8oc7836)c%2(
1%2-490%8)c-2*361%8-32ck-q)qoc(%8%1q
```

وهذه بالطبع هى الصيغة المشفرة للنص الذى سبق إدخاله .

افرض الآن أن المستفيد اختار الاختيار الثالث . يظهر ملقن للمفتاح Key مرة أخرى . ويجب أن يستجيب المستفيد بإدخال C ، وهو نفس المفتاح الذى استخدمه فى عمل الشفرة للنص . وبمجرد إدخال المفتاح تقرأ البيانات المشفرة مرة أخرى من ملف النص Code ، وتفك شفرتها رمزا رمزا ، ثم تعرض على النحو التالى :

```
All digital computers, regardless of their size, are
basically electronic devices that can transmit, store and
manipulate information (i.e., data).
```

وعلى هذا ... فقد أعيد تكوين النص الأصلى .

إذا ما استخدم مفتاح خاطئ وتم إدخاله كاستجابة للملقن ، فيعاد فك الشفرة للنص بطريقة خاطئة ، وتضيع معالم النص الأصلى ، ويظهر نص لامعنى له . افرض على سبيل المثال أن المستفيد أدخل B ، بدلا من C كمفتاح Key . عند ذلك يظهر النص التالى :

```
Bmm!ejhjubm!dpnqvufst-!sfhbsemftt!pg!uifjs!tj{f-!bsf
cbtjdbmmz!fmfduspojd!efwjdft!uibu!dbo!usbotnju-!tupsf!boe
nbojqvmbuf!jogpsnbujpo!)j/f/-!ebub*/
```

أخيرا ، افرض أن المستفيد اختار إنهاء الحسابات باختياره للبديل الأخير فى القائمة . عند ذلك تظهر الرسالة التالية :

```
That's all, folks!
```

وتفصل الحسابات .

## أسئلة مراجعة : Review Questions

(١) ماهى الخواص الأساسية للملف ؟ وكيف يختلف الملف عن المنظومة ؟

(٢) ماهو الفرق بين الملف الدائم والملف المؤقت ؟ ماهو الاصطلاح المستخدم للإشارة إلى الملفات الدائمة فى البسكال ؟

(٣) ماهو الفرق بين الملف الخارجى والملف الداخلى ؟

(٤) ماهو الفرق بين الملف التتابعى والملف العشوائى ؟ وما هى مميزات وعيوب كل منهما ؟

(٥) ماهو الاسم الآخر للملف الاتصال العشوائى ؟ أى الاسمين معبر أكثر ؟ وضح ذلك .

(٦) ماهو نوع الملفات المستخدم فى بسكال ISO القياسى ؟

(٧) لخص قواعد تعريف نوع الملفات .

(٨) لخص قواعد تعريف متغير من نوع ملف . قارن ذلك بإجابتك على السؤال السابق .

(٩) ماهى أنواع البيانات التى يمكن أن تصاحب مكونات ملف فردى ؟

(١٠) ماهى القيود الموضوعة على أقصى طول مسموح به للملف ؟ وهل هذه القيود موضوعة على أى نوع من أنواع البيانات المرتبة الأخرى ؟

(١١) كيف تمرر الملفات الخارجية إلى أو من برنامج البسكال ؟ وكيف يتحقق ذلك ؟

(١٢) ماهو احتياطى الملف ؟ وكيف يسمى احتياطى الملف ؟ ومانوع البيانات التى يمكن أن يمثلها ؟

(١٣) ماهى أول خطوة فى إنتاج ملف جديد ؟ وكيف يمكن تحقيق ذلك ؟

(١٤) ماهو تأثير عبارة rewrite عند إعداد الملف الموجود للكتابة ؟

(١٥) ماهو الغرض من عبارة put ؟ وكيف يمكن تكرار استخدام هذه العبارة فى كتابة مكونات فى ملف ؟

(١٦) افرض أن أحد الملفات يحتوى على مكونات مرتبة يراد إنتاجه . كيف يمكن إدخال عناصر البيانات الفردية لكل مكون من مكونات الملف داخل الكمبيوتر وكتابتها فى الملف ؟ هل يجب أن تكون كل عناصر البيانات هذه من نفس النوع ؟

(١٧) كيف يمكن تمرير إحدى مكونات ملف مرتب كمؤشر بين إجراء والمجموعة الإجرائية الرئيسية ؟

(١٨) ماهو الغرض من عبارة write ؟ وكيف تختلف هذه العبارة عن عبارة put ؟ وضح ذلك بالتفصيل .

(١٩) مانوع عناصر المخرجات التى يمكن أن توجد فى عبارة write ؟ هل يمكن وجود عناصر مخرجات متعددة فى عبارة write واحدة ؟

(٢٠) كيف يمكن إعداد ملف موجود للقراءة ؟

(٢١) ماهو تأثير عبارة reset على ملف يحتوى على مكون واحد أو أكثر ؟

(٢٢) ماهو تأثير عبارة reset على ملف فارغ ؟ قارن إجابتك بالإجابة المعطاه للسؤال السابق .

(٢٣) ماهو الغرض من عبارة get ؟ كيف يمكن تكرار استخدام هذه العبارة فى قراءة مكونات متعددة من ملف ؟

(٢٤) متى يقرأ الملف ؟ وكيف تكتشف نهايته ؟

(٢٥) ماهو الغرض من عبارة read ؟ وكيف تختلف هذه العبارة عن عبارة get ؟ وضح ذلك بالتفصيل .

(٢٦) مانوع عناصر المدخلات التى يمكن أن تظهر فى عبارة read ؟ هل يمكن أن تظهر متغيرات متعددة فى عبارة read ؟

(٢٧) هل يمكن لبرنامج بسكال أن يقرأ معلومات من ملف ، ثم يضيف معلومات جديدة لنفس الملف ؟ وضح ذلك .

(٢٨) لخص الخطوات التى يجب اتباعها لتجديد ملف موجود .

(٢٩) هل يمكن نقل ملف إلى إجراء أو إلى دالة كمؤشر قيمة ؟ وضح ذلك .

(٣٠) ماهو ملف النص ؟ وكيف تختلف ملفات النصوص عن أنواع الملفات الأخرى ؟

(٣١) كيف تفسر الرموز الموجودة فى أحد أسطر أحد النصوص ؟

(٣٢) كيف يعرف ملف النص ؟

(٣٣) هل يمكن استخدام عبارة get وعبارة put مع ملف النص ؟

(٣٤) هل تستخدم عبارة get وعبارة put بصورة معتادة مع ملف النص ؟ وضح ذلك .

(٣٥) هل يمكن استخدام الإجراءات القياسية الموجهة للملفات ، مثل eof , rewrite , reset مع ملف النص ؟

(٣٦) ماهو الغرض من عبارة readln ؟ وكيف تختلف هذه العبارة عن عبارة read ؟ هل يمكن استخدام عبارة readln مع كل أنواع الملفات ؟ .

(٣٧) افرض أن عبارة readln , read تحتوى على متغير من النوع الصحيح أو الحقيقى . كيف تفسر الرموز المناظرة فى ملف النص ؟

(٣٨) ماهو الغرض من الدالة القياسية eoln ؟ قارن ذلك بالدالة القياسية eof .

(٣٩) ماهو الغرض من عبارة writeln ؟ وكيف تختلف هذه العبارة عن عبارة write ؟ هل يمكن استخدام عبارة writeln مع كل أنواع الملفات ؟

(٤٠) افرض أن عبارة write أو عبارة writeln تحتوى على متغير أو على تعبير من نوع بسيط ، بدلا من النوع الحرفى . كيف تفسر الرموز المناظرة فى ملف النص ؟

(٤١) ماهو الغرض من الإجراء القياسى page ؟ هل يفسر هذا الإجراء بنفس الطريقة بواسطة كل وحدات المخرجات ؟

(٤٢) اعمل تخطيطا لطريقة نقل محتويات ملف نص من أحد ملفات النصوص إلى ملف نص آخر سطرا سطرا .

(٤٣) كيف يختلف الملفان سابقى التوضيح output , input عن ملفات النصوص الأخرى ؟

(٤٤) صف التبسيطات المسموح بها عند استخدام أحد ملفات النصوص سابقة التوضيح مع إجراءات ودوال قياسية ، مثل page , eof , write , read .

(٤٥) هل عبارة reset مطلوبة عند الإعداد لقراءة ملف مدخلات ، وهل rewrite مطلوبة مع ملف مخرجات ؟

## Solved Problems                مسائل محلولة :

**(٤٦) فيما يلى تعريفات عديدة للملفات**

(*a*) 
```
VAR data : FILE OF integer;
```
(*b*) 
```
VAR sales,costs : FILE OF real;
```
(*c*) 
```
TYPE acctno = 1..9999;
VAR accounts : FILE OF acctno;
```
(*d*) 
```
TYPE color = (red,green,blue);
     sample = RECORD
                    first : color;
                    second : 1..132;
                    third : char
                 END;
VAR data : FILE OF sample;
```
(*e*) 
```
TYPE color = (red,green,blue);
     sample = RECORD
                    first : color;
                    second : 1..132;
                    third : char
                 END;
     list = ARRAY [1..100] OF sample;
VAR data : FILE OF list;
```
(*f*) 
```
VAR oldstuff,newstuff : text;
```

**(٤٧) توضح البرامج التلخيصية الآتية عمليات تقليدية للملفات :**

(*a*) Enter a sequence of records from the keyboard and save in a file.

```
PROGRAM sample(input,data);
TYPE line = PACKED ARRAY [1..80] OF char;
     personal = RECORD
                    name : line;
                    address : line;
                    phone : line
                 END;
VAR data : FILE OF personal;
    nameandaddress : personal;

PROCEDURE readline(VAR info : line);
VAR count : 1..80;
BEGIN
   FOR count := 1 TO 80 DO read(info[count]);
   readln
END;
```

```
BEGIN
   rewrite(data);
   WITH nameandaddress DO
      BEGIN
         readline(name);
         WHILE (name[1] <> 'e') AND (name[2] <> 'n')
                                AND (name[3] <> 'd') DO
            BEGIN
               readline(address);
               readline(phone);
               write(data,nameandaddress);
               readline(name)
            END
      END
END.
```

(*b*) Read a sequence ot records from a file and display.

```
PROGRAM sample(output,data);
TYPE line = PACKED ARRAY [1..80] OF char;
     personal = RECORD
                    name : line;
                    address : line;
                    phone : line
                 END;
VAR data : FILE OF personal;
    nameandaddress : personal;

BEGIN
   reset(data);
   WITH nameandaddress DO
      WHILE NOT eof(data) DO
         BEGIN
            read(data,nameandaddress);
            writeln(name);
            writeln(address);
            writeln(phone);
            writeln
         END
END.
```

(*c*) Copy a sequence of records from one file to another.

```
PROGRAM sample(data1,data2);
TYPE line = PACKED ARRAY [1..80] OF char;
     personal = RECORD
                    name : line;
                    address : line;
                    phone : line
                 END;
VAR data1,data2 : FILE OF personal;
    nameandaddress : personal;

BEGIN
   reset(data1);
   rewrite(data2);
   WHILE NOT eof(data1) DO
      BEGIN
         read(data1,nameandaddress);
         write(data2,nameandaddress)
      END
END.
```

(*d*) Copy a sequence of records from one file to another, using get and put.

```
PROGRAM sample(data1,data2);
TYPE line = PACKED ARRAY [1..80] OF char;
     personal = RECORD
                   name : line;
                   address : line;
                   phone : line
                END;
VAR data1,data2 : FILE OF personal;
    nameandaddress : personal;

BEGIN
   reset(data1);
   rewrite(data2);
   WHILE NOT eof(data1) DO
      BEGIN
         nameandaddress := data1↑;
         get(data1);
         data2↑ := nameandaddress;
         put(data2)
      END
END.
```

(*e*) Enter a name from the keyboard. Then search a file for a record with the same name.

```
PROGRAM sample(input,data);
TYPE line = PACKED ARRAY [1..80] OF char;
     personal = RECORD
                   name : line;
                   address : line;
                   phone : line
                END;
VAR count : 1..80;
    newname : line;
    data : FILE OF personal;
    nameandaddress : personal;

BEGIN
   reset(data);
   FOR count := 1 TO 80 DO read(newname[count]);
   REPEAT
      read(data,nameandaddress)
   UNTIL nameandaddress.name = newname
END.
```

(*f*) Enter a sequence of names, addresses and phone numbers and save in a text file.

```
PROGRAM sample(input,data);
TYPE line = PACKED ARRAY [1..80] OF char;
VAR name,address,phone : line;
    data : text;

PROCEDURE readline(VAR info : line);
VAR count : 1..80;
BEGIN
   FOR count := 1 TO 80 DO read(info[count]);
   readln
END;
```

```
BEGIN
   rewrite(data);
   readline(name);
   WHILE (name[1] <> 'e') AND (name[2] <> 'n')
                          AND (name[3] <> 'd') DO
      BEGIN
         readline(address);
         readline(phone);
         write(data,name,address);
         writeln(data,phone);
         readline(name)
      END
END.
```

## Supplementary Problems — مشاكل متكاملة :

(٤٨) التخطيطات الهيكلية التالية توضح حالات عديدة مختلفة ، تشمل استخدام ملفات ، وبعضها مكتوب بطريقة خاطئة . حدد كل الأخطاء .

(*a*)
```
TYPE olddata,newdata : FILE OF integer;
```

(*b*)
```
VAR flags : FILE OF boolean;
```

(*c*)
```
VAR story : FILE OF text;
```

(*d*)
```
TYPE status = (current,overdue,delinquent);
     account = RECORD
                  custname : PACKED ARRAY [1..80] OF char;
                  custno : 1..9999;
                  custtype : status;
                  custbalance : real
               END;
     listing = ARRAY [1..1000] OF account;
     posting = FILE OF listing;
VAR oldposting,newposting : posting;
```

(*e*)
```
TYPE color = (red,green,blue);
     sample = RECORD
                 first : color;
                 second : 1..132;
                 third : char
              END;
     data = FILE OF sample;
VAR olddata,newdata : FILE OF data;
```

(*f*)
```
VAR input,output : text;
```

(*g*)
```
PROGRAM sample(input,data);
VAR data : FILE OF real;
    item : real;
    BEGIN
       readln(item);
       WHILE item <> 0 DO
          BEGIN
             writeln(data,item);
             readln(item)
          END
    END.
```

(*h*)
```
PROGRAM sample(file1,file2);
VAR file1,file2 : text;
    byte : char;

BEGIN
   reset(file2);
   rewrite(file1);
   WHILE NOT eof(file2) DO
      BEGIN
         WHILE NOT eoln(file2) DO
            BEGIN
               read(file2,byte);
               write(file1,byte)
            END;
         readln(file2);
         writeln(file1)
      END
END.
```

(i)
```
PROGRAM sample(input,output,posting);
TYPE line = PACKED ARRAY [1..80] OF char;
     account = RECORD
                   custname : line;
                   custno : 1..9999;
                   custbalance : real
               END;
VAR name : line;
    count : 1..80;
    customer : account;
    posting : FILE OF account;
BEGIN
   reset(posting);
   customer := posting;
   FOR count := 1 TO 80 DO read(name[count]);
   WHILE customer.custname <> name DO
      BEGIN
         get(posting);
         customer := posting
      END;
   WITH customer DO
      writeln(custname,custno,custbalance)
END.
```

(j)
```
PROGRAM sample(input,output,oldaccts,newaccts);
TYPE line = PACKED ARRAY [1..80] OF char;
     account = RECORD
                   custname : line;
                   custno : 1..9999;
                   custbalance : real
               END;
     filetype = FILE OF account;
VAR oldaccts,newaccts : filetype;

PROCEDURE rearrange(oldfile,newfile : filetype);
BEGIN
   .
   .
   (* rearrange the records by ascending customer number *);
   .
   .
END;

BEGIN  (* main action statements *)
   .
   .
   rearrange(oldaccts,newaccts);
   .
   .
END.
```

## مشاكل برمجة : Programming Problems

(٤٩) اكتب برنامجا كاملا بلغة البسكال ، ينتج ملف البيانات القديم الموضح فى مثال ١١ - ٢٠ . نفذ البرنامج منتجا ملف بيانات يستخدم فى المشكلة التالية :

(٥٠) عدل برنامج فواتير العملاء المعطى فى مثال ١١ - ٢٠ بإضافة المعالم التالية :

أ - حدد أن كل سجل يجدد بإدخال رقم العميل من لوحة مفاتيح ، بدلا من محاولة تجديد السجلات التى لها موازنة حالية قائمة ( انظر تخطيط البرنامج الموضح فى مثال ١١ - ١٩ ) .

ب - أدخل جزءا لزيادة الموازنة القائمة ( أى إضافة دين جديد ) أثناء إعداد الفواتير لهذه الفترة . حدد مرة أخرى السجلات التى ستجدد بهذه الطريقة عن طريق إدخال أرقام العملاء المناسبة من لوحة المفاتيح .

استخدم البرنامج فى تشغيل ملف البيانات الناتج من آخر برنامج ، مع كتابة الديون الجديدة التالية :

| Customer | New charge |
|---|---|
| Alan B. Adams | 245.00 |
| William D. McDonald | 88.50 |

(٥١) عدل برنامج عمل الشفرة وفكها الموجود فى مثال ١١ - ٢٤ بحيث يمكن إدخال مفتاح key متعدد الأرقام مع استخدام أرقام متتالية لكل سطر متتال . فمثلا اذا ما أدخل مفتاح ذو ثلاثة أرقام ، فيستخدم أول رقم لعمل الشفرة وفكها لأول سطر . والرقم الثانى للسطر الثانى ، والرقم الثالث للسطر الثالث . فإذا ماكان هناك أسطر أكثر من الأرقام الموجودة على المفتاح ، فتستخدم نفس هذه الأرقام مرة أخرى بنفس هذا الترتيب ، أى يستخدم الرقم الأول للسطر الرابع ، والرقم الثانى للسطر الخامس ، والرقم الثالث للسطر السادس .

اختبر البرنامج مستخدما العديد من الأسطر التى تختارها بنفسك .

(٥٢) عدل من محاكاة المباراة الموجودة فى مثال (٨ -١٨) ، بحيث تحاكى عدداً محدداً من المباريات وتحفظ ( وتخزن ) ناتج كل مباراة فى ملف نصوص . وفى نهاية المحاكاه اقرأ ملف النصوص لتحديد نسبة الكسب والخسارة للاعب .

اختبر البرنامج بمحاكاة ١٠٠ مباراة متتالية . استخدم النتائج فى تقدير نسبة المكسب .

(٥٣) عدل برنامج إنتاج pig latin الموجود فى مثال ( ٩ -٢٧) ، بحيث يمكن إدخال أسطر عديدة من النص عن طريق لوحة المفاتيح . خزن محتوى النص الإنجليزى فى ملف نصوص ، وخزن المحتوى المناظر له - والمكتوب بـ pig latin - فى ملف نصوص آخر .

أدخل فى البرنامج جزءا لإنتاج قائمة تسمح للمستفيد باختيار أى سمة من السمات التالية :

أ - إدخال نص جديد ، وتحويله إلى pig latin وتخزينه ( يتم تخزين كل من الملف الأصلى وملف pig latin ) .

ب - قراءة النص الذى سبق إدخاله من ملف النصوص وعرضه .

ج – قراءة pig latin المناظر للنص الذى سبق إدخاله وعرضه .

د – إنهاء الحسابات .

اختبر البرنامج ، مستخدما أسطر اختيارية من أحد النصوص .

(٥٤) عدل نظام مراقبة المخزون المذكور فى مثال ( ١٠ – ٢٨ ) ، بحيث إنه يشمل ملف بيانات يحتوى على سـجلات فردية . أدخل جزءا فى البرنامج لأداء أى عملية من العمليات التالية :

أ – إضافة سجل جديد .

ب – تعديل سجل موجود ( بما فى ذلك تعديلات فى المعلومات الوصفية ، أو فى ضبط كميات المخزون ) .

ج – حذف سجل .

د – إنتاج قائمة كاملة لكل العناصر الموجودة فى المخزون .

هـ – إنهاء الحسابات .

اسمح باختيار العمليات الفردية من قائمة . اختبر البرنامج مستخدما بيانات العينة الموجودة فى مثـال ( ١٠ – ٢٨ ) .

(٥٥) اكتب برنامجا كاملا بلغة البسكال ينتج ملف بيانات يحتوى على بيانات امتحانات الطلبة الموجود فى المشكلة رقم ( ٥٢ من الفصل التاسع ) . اجعل كل مكون من مكونات الملف يكون سجلا يحتوى على الاسم ودرجات الامتحان لكل طالب . نفذ البرنامج منتجا ملف بيانات لاستخدامه فى المشكلة التالية .

(٥٦) اكتب برنامجا بلغة البسكال موجها للملفات ، يقوم بتشغيل درجات امتحانات الطلبة المعطاه فى المشكلة رقم ( ٥٢ من الفصل التاسع ) . اقرأ البيانات من ملف البيانات الذى أعد فى المشكلة السابقة . بعد ذلك أنتج تقريرا يحتوى على الاسم ودرجات الامتحان ومتوسط الدرجة لكل طالب .

(٥٧) وسع البرنامج المكتوب فى المشكلة السابقة لتحديد متوسط شامل لدرجة الفصل ، يتبعه انحراف متوسط كل طالب عن متوسط الفصل . اكتب المخرجات فى ملف بيانات جديد . بعد ذلك أعرض المخرجات على هيئة تقرير به عناوين جيدة للتوضيح .

(٥٨) اكتب برنامجا متداخلا موجها للملفات يحفظ قائمة بأسماء وعناوين وأرقام الهاتف مرتبة ترتيبا أبجديا ( طبقا لإسم العائلة ) . ضع المعلومات المصاحبة لكل اسم فى سجل منفصل . يحتوى البرنامج على قائمة تسمح للمستفيد بأن يختار أى عملية من العمليات التالية :

أ – إضافة سجل جديد .

ب – حذف سجل .

ج – تعديل سجل موجود .

د – استرجاع وعرض محتوى سجل يناظر أسمًا معينًا .

هـ – إنتاج قائمة كاملة بكل الأسماء والعناوين وأرقام التليفونات .

و – إنهاء الحسابات .

تأكد من إعادة ترتيب السجلات عند إضافة سجل ، أو عند حذف سجل بحيث تظل السجلات مرتبة ترتيبا أبجديا بصفة دائمة .

(٥٩) اكتب برنامجا ينتج ملف بيانات يحتوى على قائمة بالبلاد وعواصمها ، والموجود فى المشكلة ( ٥٦ من الفصل التاسع ) . ضع اسم كل بلد وعاصمتها فى سجل منفصل . نفذ البرنامج منتجا ملف بيانات لاستخدامه فى المشكلة التالية .

(٦٠) اكتب برنامجا متداخلا يعمل بواسطة القائمة يتصل بملف البيانات الذى ينتج من المشكلة السابقة ، ويسمح بأداء إحدى العمليات التالية :

أ – تحديد عاصمة بلد محدد .

ب – تحديد البلد المعروف عاصمته .

ج – إنهاء الحسابات .

(٦١) وسع البرنامج المكتوب فى المشكلة السابقة ليشمل المعالم الإضافية التالية :

أ – إضافة سجل جديد .

ب – حذف سجل .

تأكد أن قائمة البلاد تحفظ بالترتيب الأبجدى عند إضافة أو حذف أى سجل .

(٦٢) اكتب برنامجا كاملا بلغة البسكال يمكن استخدامه كمنقح بسيط موجه للأسطر . يجب أن يكون لهذا البرنامج المقدرات التالية :

أ – إدخال العديد من الأسطر من أحد النصوص ، وتخزينها فى ملف نصوص .

ب – سرد ملف النصوص .

ج – استرجاع وعرض سطر معين يتم تحديده عن طريق رقم السطر .

د – إدخال n سطرا .

هـ – حذف n سطرا .

و – حفظ النص الجديد المنقح ، وإنهاء الحسابات .

يجب أن يؤدى كل نشاط من هذه الأنشطة ، كاستجابة لأمر عبارة عن إدخال حرف واحد ، تسبقه علامة دولار . ويجب أن يتبع أمر الاسترجاع رقم صحيح بدون إشارة ليحدد السطر المطلوب استرجاعه . كما يمكن أن يلى أمر الحذف وأمر الإضافة رقم صحيح ، بدون إشارة إختياريا ، إذا ما كان مطلوبًا حذف أو إضافة عدة أسطر متتالية .

ويجب أن يظهر كل أمر من هذه الأوامر فى سطر مستقل به ، وذلك ليقدم وسيلة لتمييز الأوامر عن أسطر النص . ( يبدأ سطر الأمر بعلامة دولار ، يتبعها أمر مكون من حرف فردى ، ثم يلى ذلك رقم صحيح بدون إشارة (اختيارى) ومحدد لنهاية السطر ) .

ونوصى باستخدام الأوامر التالية :

$E إدخال نص جديد

$L سرد محتويات النص

$Fk استرجاع السطر رقم k .

$In إدخال n سطرًا بعد السطر رقم k .

$Dn حذف n سطرًا بعد السطر رقم k .

$S حفظ النص المنقح ، وإنهاء الحسابات .

(٦٣) وسع البرنامج المذكور فى المشكلة ( ٤٢ من الفصل العاشر ) ، بحيث تحفظ معلومات الفرق فى ملف بيانات ، بدلا من حفظها فى منظومة . ويجب أن يصبح كل مكون من مكونات الملف سجلا محتويا على بيانات إحدى الفرق . اجعل البرنامج محتويا على أجزاء تؤدى مايلى :

أ – إدخال سجلات جديدة ( إضافة فرق جديدة ) .

ب – تجديد سجلات موجودة .

ج – حذف سجلات ( إلغاء فرق ) .

د – إنتاج تقرير تلخيصى لكل الفرق .

# الفصل الثانى عشر

# الفئات

# Sets

تعرف الفئة set رسميا فى البسكال بأنها مجموعة من عناصر البيانات البسيطة المرتبة ، والتى لها نفس النوع . وعلى هذا ... يمكن أن تكون الفئة عبارة عن مجموعة من عناصر البيانات الصحيحة ، أو عناصر البيانات الحرفية ، أو عناصر البيانات المتعددة .

ولكى يستخدم مفهوم الفئة ، يجب أن نعرف أولا نوع الفئة . ويمكننا عند ذلك توضيح متغيرات من نوع الفئة تكون قيمتها الفردية عبارة عن عناصر من هذا النوع من أنواع الفئات . وفى واقع الأمر ... يمكن لمتغير واحد من نوع الفئة أن يمثل أى عدد من عناصر الفئة بما فيها العنصر الفارغ . وتقدم هذه الإمكانية طريقة بسيطة لنا لتحديد إذا ماوقع حدث أو كينونة entity فى إحدى الفئات أو أكثر من فئة سبق تعريفها .

## ١ – تعريف نوع الفئة 1. DEFINING A SET TYPE

نبدأ بمصاحبة مجموعة من عناصر البيانات البسيطة النوع والمرتبة ، مستخدمين تعريف TYPE كما سبق أن فعلنا ذلك . وهذا النوع من البيانات يعرف بأنه النوع الأساسى base type . وعلى هذا ... يمكننا تحديد النوع الأساسى بأنه :

```
TYPE base type = (data item 1, data item 2, . . . ,data item n)
```

أو :

```
TYPE base type = first data item .. last data item
```

ونوع الفئة الذى نريد تعريفه ، مقدم على ذلك بالنسبة للنوع الأساسى ، أى أن :

```
set type = SET OF base type
```

وعلى هذا ... فإن نوع الفئة يشير إلى نفس مجموعة عناصر البيانات مثل نوع الأساس .

وبعد تعريف نوع الفئة ، فإننا نوضح متغير نوع الفئة بالطريقة التالية :

```
VAR set name : set type
```

أو ، إذا كان مطلوبا متغيرات متعددة مختلفة من نوع الفئة ، فتوضح كمايلى :

```
VAR set name 1, set name 2, . . . ,set name n : set type
```

يمكن أن تمثل هذه المتغيرات من نوع الفئة بعض الفئات الجزئية للعناصر الموجودة فى النوع الأساسى .

## مثـال (١٢-١)

اعتبر التوضيح التالى :

```
TYPE sizes = (small,medium,large);
     shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
```

فى هذا المثال sizes هى نوع أساسى يحتوى على عناصر بيانات متعددة long و medium و small . نوع الفئة هو shirtsizes . ( لاحظ أن shirtsizes معرف بالنسبة للنوع الأساسى sizes ) . أخيرا فإن longsleeve , shortsleeve متغيران من نوع الفئة لهما نفس النوع shirtsizes .

ويمكننا إذا رغبنا أن نعرف أنواع فئات مختلفة عديدة من نفس النوع الأساسى . وعلى أية حال ... معظم التطبيقات البسيطة لاتحتاج إلى مثل هذا التعقيد .

## مثـال (١٢-٢)

فيما يلى صيغة مختلفة من تعريفات نوع الفئة المعطاة فى مثال (١ - ١٢) .

```
TYPE sizes = (small,medium,large);
     shirtsizes,dresssizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
    shorthem,longhem : dresssizes;
```

لاحظ أن كلا من dresssizes , shirtsizes نوعى بيانات من نفس النوع الأساسى sizes . لاحظ أيضا أن longsleeve , shortsleeve متغيران من نوع الفئة ، وأن longhem , shorthem متغيران من نوع الفئة ، لهما نفس النوع مثل dresssizes .

يمكننا أيضا أن نعرف نوع فئة بالنسبة لنوع بسيط ومرتب وقياسى ( وهو النوع الصحيح والنوع الحرفى ) أو نوع المدى الجزئى لنوع بسيط ومرتب وقياسى . فى مثل هذه الحالات يصبح نوع البيانات القياسى أو المدى الجزئى منه نوعا أساسيا .

## مثـال (١٢-٣)

كل نوع من أنواع الفئات التالية معرف بالنسبة لنوع بسيط ، ومرتب ، وقياسى ، أو بالنسبة للمدى الجزئى المناظر .

```
TYPE numbers = SET OF integer;

TYPE digits = SET OF 0..9;

TYPE numchars = SET OF '0'..'9';

TYPE lowercase = SET OF 'a'..'b';
```

لاحظ أن ثانى نوع فئة يعرف فئات من النوع الصحيح ، بينما النوع الثالث يعرف فئات من النوع الحرفى .

## ٢ – عمل الفئة 2. CONSTRUCTING A SET

دعنا نوجه انتباهنا الآن إلى عمل فئات فردية . يمكن أن تحدد مثل هذه الفئات لمتغيرات من نوع الفئة مناظرة لها . كما يمكن استخدامها أيضا كمؤشرات فى بعض أنواع تعبيرات بوليان ( سيذكر المزيد عن ذلك فيما بعد ) .

يمكن أن تحتوى الفئة على أى عدد من العناصر من الفئة الأساسية المصاحبة . وتعد الفئة بكتابة العناصر الفردية على التوالى محصورة بين قوسين مربعين ، ومفصولة عن بعضها بواسطة فواصل . وعلى هذا ... تبدو الفئة الفردية على النحو التالى :

[*set element 1*, *set element 2*, . . . , *set element n*]

العناصر المحتواه تعرف بأنها أعضاء members فى الفئة .

ويمكن أن تحتوى الفئة بالطبع على عنصر واحد ، كما أنه من الممكن أيضا إعداد فئة لاتحتوى على أى عنصر على الإطلاق . وهذه تعرف بأنها فئة صفرية null أو فئة فارغة empty . وتكتب على النحو التالى : [ ]

### مثال (١٢–٤)

فيما يلى عدة فئات يمكن إعدادها من النوع الأساسى sizes المعرف فى مثال (١٢– ١ )

```
[small,medium,large]
[medium,large]
[large,small]
[medium]
[]
```

لاحظ أن أعضاء الفئة لاتحتاج إلى أن تكون مرتبة ، كما هو موضح بالفئة الثالثة . لاحظ أيضا أن آخر فئة هى فئة خالية .

بعض أعضاء الفئة يمكن تمثيلها بواسطة متغيرات ، على أن تمثل هذه المتغيرات عناصرا من نوع مناسب للنوع الأساسى . والأكثر من ذلك ... إذا ماكان بعض أعضاء الفئة عناصر متتالية ، فيمكن تمثيلها كمدى جزئى على النحو التالى :

*first consecutive element* .. *last consecutive element*

### مثال (١٢–٥)

فيما يلى فئات إضافية أعدت من النوع الأساسى sizes ، وعدة متغيرات مناظرة لها .

```
TYPE sizes = (small,medium,large);
     shirtsizes = SET OF sizes;
VAR shirt,blouse : sizes;

[small..large]

[shirt]

[shirt,blouse]

[medium,large,blouse]
```

( لاحظ أن المتغيرات المستخدمة فى هذا المثال هى متغيرات من النوع البسيط . وهى ليست متغيرات من نوع الفئة ، كما سبق ذكره فى القسم السابق ) .

وعنصر الفئة لايوجد فى نفس الفئة إلا مرة واحدة فقط ، إلا أنه – على أية حال – من الممكن أن يحدد عنصر واحد مرتين أو أكثر إذا ما احتوى وصف الفئة على عناصر فئة صريحة ومتغيرات . ( يمكن أن يمثل المتغير عنصر فئة تم تحديده بالفعل ينتج عن ذلك ازدواج غير مرغوب فيه ) . فى مثل هذه الحالات يهمل الوصف الزائد .

## مثـال (١٢-٦)

اعتبر الفئة التالية :

[medium,large,blouse]

والموجودة فى آخر مثال . تذكر أن medium , large هما عنصران من النوع الأساسى sizes . أما blouse فهو متغير variable من نفس نوع sizes . إذا ماتحدد أى من medium أو large للمتغير blouse ، فسوف يتكرر هذا العنصر داخل الفئة . وتفسر على ذلك الفئة بأن لها عضوين أثنين فقط هما medium , large . فإذا مامثلت blouse العضو small ، فإن الفئة يصبح بها ثلاثة أعضاء ، وهى small , medium , large .

ويمكن ظهور مشاكل شبيهه إذا ماعبر عن بعض عناصر الفئة بأنها من نوع المدى الجزئى . فإذا كان العنصر الأول والعنصر الأخير من المدى الجزئى هما نفس العنصر ، فيفسر المدى الجزئى بأنه عنصر واحد . وأكثر من ذلك ... إذا ما اختلف العنصر الأول عن العنصر الأخير ... ولكن بالترتيب الخاطئ ( أى إذا ماجاء العنصر الأول بعد العنصر الأخير ) ، تعتبر الفئة فارغة .

## مثـال (١٢-٧)

اعتبر الآن الفئة التالية :

[shirt .. blouse]

حيث shirt , blouse متغيران من نفس نوع sizes كما سبق توضيحهما فى مثال (٥ – ١٢) . اذا مامثل shirt , blouse نفس عنصر النوع الأساسى ( أى medium )، فإن الفئة تحتوى على عضو واحد فقط ، وهو العضو medium . كذلك إذا مامثل shirt عنصر يأتى بعد blouse فى النوع الأساسى ( أى إذا ماكان shirt يمثل large ، وكان blouse يمثل small ) ، فتعتبر الفئة خالية .

ويعد إعداد الفئة يمكن أن تحدد لمتغير من نوع الفئة . ويمكن تحقيق ذلك بالطريقة المعتادة بكتابة مايلى :

*variable name* := [*set element 1*, *set element 2*, . . . , *set element n*]

يجب أن يكون مفهوما أن الفئة التى تظهر فى الناحية اليمنى يشار إليها بأنها عنصر بيانات أحادى القيمة .

ويجب أن يكون عنصر البيانات هذا من نفس نوع الفئة المتغير الذى يحدد له .

### مثـال (١٢-٨)

التخطيط الهيكلى التالى يوضح تحديد فئة داخل برنامج بسكال .

```
PROGRAM sample(input,output);
TYPE sizes = (small,medium,large);
     shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
BEGIN
   .
   .
   shortsleeve := [small,large];
   .
   .
   longsleeve := [small,medium,large];
   .
   .
END.
```

## ٣ - العمليات التى تجرى على الفئات 3. OPERATIONS WITH SETS

هناك ثلاث عمليات مختلفة يمكن إجراؤها على الفئات ، وينتج عنها وجود فئة جديدة ونشير الى ناتج هذه العمليات ( أى الفئات الجديدة ) بأنها اتحاد union ، أو تقاطع intersection ، أو فرق الفئة set difference ، وذلك للفئتين الأصليتين على التوالى .

وتتطلب كل عملية عاملين ( أى فئتين ) من نفس النوع . والنتيجة على هذا تكون من نفس نوع العاملين .

واتحاد فئتين هو فئة جديدة تحتوى على كل أعضاء الفئتين الأصليتين ، ويستخدم المؤثر + لتحديد هذه العملية كما هو موضح أدناه .

### مثـال (١٢-٩)

يوضح هذا المثال اتحاد فئتين .

```
PROGRAM sample(input,output);
TYPE sizes = (small,medium,large);
     shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
BEGIN
   .
   .
   shortsleeve := [small] + [large];
   .
   .
   longsleeve := [small,medium] + [small,large];
   .
   .
END.
```

تتسبب أول عبارة تحديد فى اتحاد الفئتين [ large ] , [ small ] وتحديدها لمتغير من نوع السلسلة shortsleeve . وعلى هذا ... فإن shortsleeve يمثل الفئة ( [ small , large ] ) .

وبالمثل تتسبب عبارة التحديد الثانية فى أن المتغير من نوع الفئة longsleeve يمثل الفئة , [ small]) . medium , large ] )

وتقاطع فئتين عبارة عن فئة ، كل أعضائها تكون أعضاء مشتركة فى كل من الفئتين الأصليتين . ويستخدم المؤثر * فى تحديد هذه العملية .

## مثال (١٢-١٠)

فيما يلى تقاطع لفئتين :

```
PROGRAM sample(input,output),
TYPE sizes = (small,medium,large);
     shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
BEGIN
   .
   .
   shortsleeve := [small,medium] * [medium,large];
   .
   .
   longsleeve := [small] * [medium,large];
   .
   .
END.
```

تتسبب أول عبارة تحديد فى تقاطع الفئتين ,([ medium , large ]) ([ small , medium ]) ، وتحديد هذا التقاطع للمتغير من نوع الفئة shortsleeve . وعلى هذا .. فإن shortsleeve يمثل الفئة [ medium ] .

وبالمثل تتسبب عبارة التحديد الثانية فى أن يمثل المتغير من نوع الفئة longsleeve الفئة الخالية [ ] ، حيث إن العاملين لايحتويان على أعضاء مشتركين .

والفرق بين فئتين ، هو فئة توجد أعضاؤها فى الفئة الأولى ولاتوجد فى الفئة الثانية . ويرمز لهذه العملية بالمؤثر - كما يتضح من المثال التالى :

## مثال (١٢-١١)

هذا المثال يوضح الفرق بين فئتين .

```
PROGRAM sample(input,output);
TYPE sizes = (small,medium,large);
     shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
BEGIN
   .
   .
   shortsleeve := [small,medium] - [small,large];
   .
   .
   longsleeve := [small,medium,large] - [medium];
   .
   .
END.
```

تتسبب أول عبارة تحديد فى حساب الفرق بين الفئتين ( [ small , medium ]) , ( [ small , large ] ) وتحديدها لمتغير من نوع الفئة shortsleeve . وعلى هذا ... يمثل shortsleeve الفئة [ medium ] .

وبالمثل تتسبب عبارة التحديد الثانية فى تحديد الفئة ( [ small , large ]) لمتغير من نوع الفئة ، وهو longsleeve .

وعادة ماتدمج هذه العمليات مع عبارات تحديد الفئات لتعديل قيم متغيرات من نوع الفئة . وهذا صحيح بصفة خاصة مع عمليات الاتحاد والفرق .

**مثـال (١٢ - ١٢)**

فيما يلى عدة عبارات تحديد تحتوى على استخدام لعمليات الفئات . يفترض أن المتغيرات , longsleeve shortsleeve من نوع الفئة كما سبق تعريفهما فى الأمثلة السابقة .

```
shortsleeve := shortsleeve + [medium];

longsleeve := longsleeve - [small];

shortsleeve := longsleeve + [large];

shortsleeve := longsleeve * [small,medium];
```

تتسبب العبارة الأولى فى إضافة العنصر medium إلى الفئة الممثلة بواسطة متغير الفئة shortsleeve . ( فإذا كان medium عضوا فى الفئة ، فلن يكون لهذه العبارة أى تأثير على الفئة ) .

والغرض من العبارة الثانية هو حذف العنصر small من الفئة التى يمثلها المتغير longsleeve ( إلا إذا لم يكن small موجودا من الأصل . وفى هذه الحالة لايكون لهذه العبارة أى تأثير ) .

وفى العبارة الثالثة يضاف large إلى الفئة التى يمثلها ، وتحدد الفئة الجديدة للمتغير shortsleeve ( لاحظ أن longsleeve لم يتغير ) .

وأخيرا تتسبب آخر عبارة فى تحديد الأعضاء المشتركين فى كل من الفئة التى يمثـلها longsleeve والفـئة [ small , medium ] للمتغير shortsleeve .

## ٤ - مقارنة الفئات : 4. SET COMPARISONS

يمكن استخدام أربعة مؤثرات من السبعة مؤثرات العلاقية مع الفئات ، وذلك لتكوين تعبيرات من نوع بوليان . وهذه المؤثرات الأربعة وتفسيراتها عند استخدامها مع الفئات هى كما يلى :

| المـؤثـر | تفسيره |
| --- | --- |
| = | تساوى الفئة ( أى أن كل فئتين تحتويان على نفس الأعضاء ، وبنفس الترتيب ) . |
| <> | عدم تساوى الفئة ( لاتحتوى الفئتان على نفس الأعضاء بالضبط ) . |

=> احتواء الفئة ( كل عضو فى الفئة الأولى موجود داخل الفئة الثانية ) .

=< احتواء الفئة ( كل عضو فى الفئة الثانية موجود فى الفئة الأولى ) .

عند استخدام أى مؤثر من هذه المؤثرات ، فيجب أن تكون الفئتان من نفس النوع .

## مثـال (١٢-١٣)

فيما يلى عدة تعبيرات بوليان تحتوى على فئات . ( افرض أن أعضاء الفئة هى عناصر من نوع البيانات المتعدد ) .

```
sizes = (small,medium,large)
```

والتى سبق تقديمها فى مثال (١٢- ١ )

| القيمة | التعبير |
|---|---|
| خطأ | [small,large] = [small,medium,large] |
| صحيح | [small,large] = [large,small] |
| صحيح | [small,medium,large] = [small..large] |
| صحيح | [small,medium] <> [medium] |
| صحيح | [small] <= [small..large] |
| خطأ | [small,medium] <= [small,large] |
| خطأ | [small..large] <= [large] |
| صحيح | [] <= [small..large] |
| صحيح | [small,medium,large] >= [medium,large] |
| صحيح | [medium,large] >= [medium,large] |
| خطأ | [medium] >= [small,medium] |

لاحظ أن الفئة الصفرية محتواه داخل أى فئة أخرى . وعلى هذا ... فإن تعبيراً مثل التعبير التالى يكون دائما صحيحا true .

```
[] <= [small,large]
```

والمؤثرات العلاقية => ، =< لها تفسيرات مختلفة بعض الشئ عند استخدامها مع المجموعات عن استخدامها مع العناصر الأخرى . فإذا كانت كل من S1 , S2 فئة ، فمن الممكن أن تكون التعبيرات S1 < = S2 , S1 > = S2 خاطئة لبعض قيم عناصر هاتين الفئتين . وهذا لايمكن حدوثه مع أنواع العناصر الأخرى .

## مثـال (١٢-١٤)

افرض أن لدينا فئتين ( مستقلتين ) mutually exclusive ، مثل [ small ] , [ large ] . تعبيرات البوليان التالية :

```
[small] <= [large]
```

```
[small] >= [large]
```

خاطئة false .

عند إجراء المقارنات بين الفئات ، يمكن التعبير عن عناصر الفئة كمتغيرات ، يحتوى نوعها الأساسى على أعضاء الفئة . أكثر من ذلك ... يمكن إجراء المقارنة بين فئة ومتغير من نوع الفئة له نفس نوع الأساس .

## مثـال (١٢-١٥)

اعتبر التخطيط الهيكلى التالى :

```
PROGRAM sample(input,output);
TYPE letters = SET OF char;
VAR used,unused : letters;
    alpha : char;
    line : PACKED ARRAY [1..80] OF char;
BEGIN
   .
   .
   FOR alpha := 'A' TO 'z' DO
      IF [alpha] <= used THEN write(alpha);
   .
   .
   WHILE NOT ([line[1]] <= ['E','e']) DO

      BEGIN
         .
         .
         .
      END;
   .
   .
END.
```

فى أول عبارة إجرائية ( وهى عبارة FOR - TO ) تقارن الفئة التى تحتوى على قيمة alpha مع قيمة used . لاحظ أن alpha متغير من النوع الحرفى ، بحيث إن [alpha] تكون فئة محتوية على عنصر واحد من النوع الحرفى . كما أن used متغير من نوع الفئة أيضا من النوع الأساسى الحرفى . وعلى هذا ... فنحن نقارن فئة تحتوى على عناصر لها نفس نوع الأساسى ( أى أننا نقارن فئتين ، عناصر كل منهما حرفية ) .

لاحظ أن دوره FOR - TO تعتبر كل الحروف متراوحة من A الكبيرة ( قيمتها ٦٥ فى شفرة ASC II - انظر الملحق G ) إلى a الصغيرة ( قيمتها ١٢٢ فى شفرة ASC II ) . وهذا يعنى أن كل الحروف الأبجدية موجودة فى عملية المقارنة .

فى العبارة الإجرائية الثانية ( عبارة WHILE - DO ) تقارن الفئة التى تحتوى على قيمة [1] line مـع الـفئة [ 'E' , 'e' ] . تذكر أن [1] line هو عنصر من عناصر منظومة حرفية النوع ، وعلى ذلك فإن [ [1] line ] تكون فئة محتويه على عنصر فردى من النوع الحرفى ، وتحتوى الفئة التى تقارن معها [ 'E' , 'e' ] على عضوين من النوع الحرفى أيضا . وعلى هذا ... فإننا نقارن فئتين عناصرهما من نفس نوع الأساس .

وتقدم الأمثلة التالية برنامج بسكال كاملا ، مشتملا على استخدام الفئات

مثـال (١٢-١٦)

*تحليل أحد أسطر النص* . افرض أننا نريد إدخال سطر من أسطر أحد النصوص داخل الكمبيوتر ، ثم نحدد أى الحروف الأبجدية موجودة داخله . يمكن تحقيق ذلك بسهولة باستخدام الفئات .

والطريقة هى قراءة السطر وتحليله عن طريق تحديد الحروف الموجودة فيه ، ثم كتابة كل الحروف التى نجدها فى السطر . نعتبر كل من الحروف الكبيرة والصغيرة ، ولن نأخذ فى الاعتبار أنواع الرموز الأخرى ( مثل التنقيط ) .

دعنا نقدم التوضيحات التالية :

```
TYPE letters = SET OF char;
VAR used,unused : letters;
    count : 0..80;
    alpha : char;
    line : PACKED ARRAY [1..80] OF char;
```

المتغيرات من نوع الفئة used , unused تمثل فئة الحروف المستخدمة والحروف غير المستخدمة على التوالى . كما يمثل line السطر الفعلى من النص .

بمعرفة هذه التوضيحات يمكن أن تظهر المجموعة الرئيسية للبرنامج على النحو التالى :

```
EGIN
  readinput;   (* read in a line of text *)
  used := [];
  unused := ['A'..'Z','a'..'z'];
  FOR count := 1 TO 80 DO
     IF [line[count]] <= unused THEN
        BEGIN
           used := used + [line[count]];
           unused := unused - [line[count]]
        END;
  writeoutput   (* write out the results of the analysis *)
END.
```

تشير العبارتان writeoutput , readinput إلى إجراءات تقرأ السطر وتكتب نتائج التحليل على التوالى :

وبعد إدخال السطر ، توضع قيم ابتدائية لمتغيرى الفئة used , unused ، وذلك بتحديد فئة صفرية للمتغير used ، وفئة تحتوى على كل الحروف الكبيرة والصغيرة للمتغير unused . ثم يمكننا بعد ذلك فحص السطر حرفا حرفا . فإذا كان الحرف غير مستخدم ( أى إذا كانت الفئة [ line [count] ] محتواه داخل الفئة unused ) ، فيحدث تجديد لكل من used , unused بإضافة الحرف الحالى إلى used ، وحذفه من unused . ويحدث هذا التجديد عن طريق كتابة مايلى :

```
used := used + [line[count]];
unused := unused - [line[count]];
```

ثم تكتب بعد ذلك نتيجة هذا التحليل

وتعمل هذه الطريقة بصورة جيدة بالنسبة لسطر واحد من أسطر النص ، إلا أنه يجب إعادة بدء البرنامج فى كل مرة ينتهى فيها السطر . دعنا على ذلك نعدل المجموعة الاجرائية الرئيسية ، بحيث يمكن تكرار تنفيذها حتى تدخل الكلمة " new " فى بداية سطر جديد . وعلى هذا ... تعدل المجموعة الاجرائية الرئيسية لتأخذ الشكل التالى :

```
BEGIN
   readinput;   (* read in a line of text *)
   WHILE NOT (([line[1]] <= ['E','e']) AND
              ([line[2]] <= ['N','n']) AND
              ([line[3]] <= ['D','d'])) DO
      BEGIN
         used := [];
         unused := ['A'..'Z','a'..'z'];
         FOR count := 1 TO 80 DO
            IF [line[count]] <= unused THEN
               BEGIN
                  used := used + [line[count]];
                  unused := unused - [line[count]]
               END;
         writeoutput;   (* write out the results of the analysis *)
         readinput
      END
END.
```

يسمح مكون WHILE - DO باستمرار الحسابات حتى يدخل سطر جديد يحتوى على الحرف c فى بدايته ، والحرف n فى العمود الثانى ، والحرف d فى العمود الثالث . لاحظ أن استخدام مقارنة الفئات داخل هذا المكون تسمح بطريقة مريحة لاختيار كل من الحروف الكبيرة والحروف الصغيرة .

ويبدأ إجراء إدخال السطر readinput بوضع قيمة ابتدائية للمتغير line ، بحيث إنه يحتوى على فراغات فقط . بعد ذلك يقرأ النص الفعلى فى الأماكن الفارغة . وتستمر القراءة حتى يكتشف محدد نهاية السطر . وفيما يلى هذا الإجراء كاملا .

```
PROCEDURE readinput;
(* this procedure reads in a line of text *)
BEGIN
   FOR count := 1 TO 80 DO line[count] := ' ';
   writeln('Please enter a line of text below');
   count := 0;
   WHILE NOT eoln DO
      BEGIN
         count := count + 1;
         read(line[count])
      END;
   readln
END;
```

إجراء المخرجات writeoutput يستخدم طريقة مختلفة . فهناك تجرى دورة خلال كل الحروف الهجائية ( الكبيرة

والصغيرة ) وتكتب هذه الحروف كأعضاء فى الفئة used . ويمكن كتابة الإجراء الكامل على النحو التالى :

```
PROCEDURE writeoutput;
(* this procedure writes out an analysis of a line of text *)
BEGIN
   writeln;
   write('Letters used:');
   FOR alpha := 'A' TO 'z' DO
      IF [alpha] <= used THEN write(' ',alpha);
   writeln;
   writeln
END;
```

تذكر أن alpha متغير من النوع الحرفى . وعلى هذا ... فكل من [ alpha ] , used يمثل فئة عناصرها من النوع الحرفى ) .

وفيما يلى برنامج البسكال كاملا .

```
PROGRAM lettersused(input,output);

(* THIS PROGRAM READS A LINE OF TEXT AND
   DETERMINES WHICH LETTERS ARE PRESENT *)

TYPE letters = SET OF char;
VAR used,unused : letters;
    count : 0..80;
    alpha : char;
    line : PACKED ARRAY [1..80] OF char;

PROCEDURE readinput;
(* this procedure reads in a line of text *)
BEGIN
   FOR count := 1 TO 80 DO line[count] := ' ';
   writeln('Please enter a line of text below');
   count := 0;
   WHILE NOT eoln DO
      BEGIN
         count := count + 1;
         read(line[count])
      END;
   readln
END;

PROCEDURE writeoutput;
(* this procedure writes out an analysis of a line of text *)
BEGIN
   writeln;
   write('Letters used:');
   FOR alpha := 'A' TO 'z' DO
      IF [alpha] <= used THEN write(' ',alpha);
```

(تكملة البرنامج فى الصفحة التالية)

```
      writeln;
      writeln
   END;

   BEGIN   (* main action block *)
      readinput;
      WHILE NOT (([line[1]] <= ['E','e']) AND
                 ([line[2]] <= ['N','n']) AND
                 ([line[3]] <= ['D','d'])) DO

         BEGIN
            used := [];
            unused := ['A'..'Z','a'..'z'];
            FOR count := 1 TO 80 DO
               IF [line[count]] <= unused THEN
                  BEGIN
                     used := used + [line[count]];
                     unused := unused - [line[count]]
                  END;
            writeoutput;
            readinput
         END
   END.
```

افرض الآن أن البرنامج يستخدم فى تشغيل السطر التالى :

ينتج عن تنفيذ البرنامج الحوار التالى ( استجابات المستفيد موضوع تحتها خط ) .

```
Please enter a line of text below
Pascal is a structured programming language derived from ALGOL-60

Letters used: A G L O P a c d e f g i l m n o p r s t u v

Please enter a line of text below
end
```

## ٥ – اختبار العضوية 5. MEMPERSHIP TESTING

تحتوى لغة البسكال على مؤثر علاقى إضافى أيضا IN يستخدم فى عمل تعبيرات بوليان . ويمكن استخدام هذا المؤثر مع عناصر من نوع الفئة فقط . وهو مفيد بصفة خاصة ، حيث إنه يسمح لنا بتحديد ماإذا كانت القيمة محتواه داخل إحدى الفئات أم لا ( أى ما إذا كانت القيمة عضواً فى الفئة أم لا ) .

ويجب استخدام المؤثر IN بالطريقة التالية لإنتاج تعبيرات بوليان .

*set element* IN *set*

يكون التعبير صحيحا true إذا كان أول عنصر عضوا فى العنصر الثانى ، وخاطئا false اذا لم يتحقق ذلك . ويجب أن يناظر كل من العنصرين نفس النوع الأساسى . ويمكن أن يكون أول عنصر عنصر فئة فرديًا أو متغيرًا أو تعبيرًا يمثل عنصر فئة . أما العنصر الثانى ، فهو فئة بصفة عامة ، أو متغيرًا من نوع الفئة .

## مثـال (١٢-١٧)

اعتبر الصيغة التالية لتوضيح الفئة الموجودة فى مثال ( ١ - ١٢ )

```
TYPE sizes = (small,medium,large);
     shirtsizes = SET OF sizes;
VAR shortsleeve,longsleeve : shirtsizes;
    mysize : sizes
```

فيما يلى عدة تعبيرات بوليان توضح استخدام المؤثر IN .

```
medium IN [small,medium,large]

medium IN [small,large]

mysize IN [small,medium]

mysize IN shortsleeve
```

التعبير الأول صحيح true ، والثانى خاطئ false ، إلا أن قيمة التعبير الثالث على أية حال تعتمد على القيمة التى تحدد للمتغير mysize . فإذا مامثل mysize القيمة small أو medium ، فإن التعبير يكون صحيحا true ، وإلا فإنه يكون خاطئا false .

وبالمثل تعتمد قيمة آخر تعبير على القيم التى تحدد للمتغير mysize والمتغير shortsleeve . إذا مامثل mysize عنصرا عضوا فى الفئة التى يمثلها shortsleeve ، فإن التعبير يكون صحيحا true ، وإلا فإنه يصبح خاطئا false .

افرض على سبيل المثال أن mysize يمثل القيمة large . يكون التعبير صحيحا إذا مامثل shortsleeve ، أى فئة من الفئات .

```
[small,medium,large]

[small,large]

[medium,large]

[large]
```

إلا أن التعبير يكون خاطئا لكل القيم الأخرى من shortsleeve .

وعادة مايتكرر اختيار العضوية مع مكونات التحكم المختلفة داخل برنامج البسكال . وهذا يسمح لنا بأداء عمليات منطقية مختلفة طبقا لاختياراتنا ، وذلك عند تحقق شروط عضوية معينة من نوع الفئة فقط .

## مثـال (١٢-١٨)

اعتبر التخطيط الهيكلى التالى :

( أنظر الصفحة التالية )

```
PROGRAM sample(input,output);
TYPE letters = SET OF char;
VAR vowels,consonants : letters;
    count,vowelcount,conscount : 0..80;
    line : PACKED ARRAY [1..80] OF char;

BEGIN
   vowels := ['A','E','I','O','U','a','e','i','o','u'];
   consonants := ['A'..'Z','a'..'z'] - vowels;
   .
   .
   vowelcount := 0;
   conscount := 0;
   .
   .
   FOR count := 1 TO 80 DO
      BEGIN
         IF line[count] IN vowels
            THEN vowelcount := vowelcount + 1;
         IF line[count] IN consonants
            THEN conscount := conscount + 1
      END;
   .
   .
END.
```

تحدد أول عبارتين قيما للمتغيرين من نوع الفئة vowels , consonants . وتختبر عبارتا IF ( الموجودتان داخل مكون FOR ) لمعرفة ماإذا كانت القيمة الحالية لعنصر منظومة من النوع الحرفى line [ count ] عضوا فى الفئة التى يمثلها vowels أو التى يمثلها consonants .

ومن المفيد مقارنة هذا التخطيط مع التخطيط الهيكلى الموضح فى مثال ( ١٢ - ١٥ ) . ففى المثال الحالى تختبر لمعرفة ما إذا ماكان عنصر بيانات فردى عضوا فى فئة أم لا . وفى الناحية الأخرى فى المثال السابق ذكره نختبر ما إذا كانت فئة محتواه داخل فئة أخرى أم لا . ( لاحظ أن توافقية النوع الأساسى مطلوبة فى كل من المثالين )

والمثال التالى يبين برنامج بسكال كاملاً يستخدم اختيار العضوية .

## مثـال (١٢-١٩)

عدد *الحروف المتحركة فى السطر* . دعنا نطور برنامجا بلغة البسكال ينفذ الأنشطة التالية :

١ - إدخال سطر من أحد النصوص داخل الكمبيوتر .

٢ - تحديد إجمالى عدد الحروف ( بما فيها الفراغات والتنقيط ) داخل السطر .

٣ - تحديد إجمالى عدد الحروف المتحركة ، وإجمالى عدد الحروف ، وعدد الحروف الساكنة داخل السطر .

٤ - كتابة إجمالى عدد الحروف ، وعدد الحروف المتحركة ، وعدد الحروف الساكنة .

اكتب البرنامج ، بحيث يكرر التنفيذ حتى يتم إدخال كلمة "end" فى بداية سطر جديد ، مع السماح بإدخال end بحروف كبيرة أو حروف صغيرة .

نبدأ بتقديم التوضيحات التالية :

```
TYPE letters = SET OF char;
VAR vowels,consonants : letters;
    count,charcount,vowelcount,conscount : 0..80;
    line : PACKED ARRAY [1..80] OF char;
```

يمثل المتغيران من نوع الفئة الحروف المتحركة والحروف الساكنة فئتى الحروف المتحركة والحروف الساكنة على التوالى . وسوف توجد الحروف الصغيرة والكبيرة فى كل فئة من الفئين . وتمثل المتغيرات الصحيحة , conscount vowelcount , charcount إجمالى عدد الرموز الموجودة فى السطر ( بما فى ذلك الفراغات والتنقيط ) وعدد الحروف المتحركة وعدد الحروف الساكنة على التوالى . وأخيرا فإن المنظومة الحرفية line تمثل السطر الفعلى من النص .

دعنا نعتبر الآن المجموعة الإجرائية الرئيسية ، إذا ماأدخلنا جزءا فى البرنامج لتكرار التنفيذ ، فيمكن كتابة المجموعة الرئيسية كما يلى :

```
BEGIN
   vowels := ['A','E','I','O','U','a','e','i','o','u'];
   consonants := ['A'..'Z','a'..'z'] - vowels;
   readinput;
   WHILE NOT ((line[1] IN ['E','e']) AND
              (line[2] IN ['N','n']) AND
              (line[3] IN ['D','d'])) DO
      BEGIN
         vowelcount := 0;
         conscount := 0;
         FOR count := 1 TO 80 DO
            BEGIN
               IF line[count] IN vowels
                  THEN vowelcount := vowelcount + 1;
               IF line[count] IN consonants
                  THEN conscount := conscount + 1
            END;
         writeoutput;
         readinput
      END
END.
```

تحدد أول عبارتين قيم المتغيرات من نوع الفئة consonants , vowel على التوالى . وتستخدم هذه القيم كنمطيات لاختبار العضوية . وعلى هذا ... فلن تتغير أثناء تنفيذ البرنامج .

ونرى عبارة readinput تلى عبارتى تحديد الفئتين . وتشير هذه العبارة إلى إجراء ، يتسبب فى قراءة سطر من النص داخل الكمبيوتر ، وعد العدد الإجمالى للرموز .

اعتبر الآن مكون WHILE - DO ، وهو يقع فى قلب البرنامج فى الواقع . يبدأ المكون بتحديد قيمة ابتدائية صفر للعدادات conscount , vowelcount . وبعد ذلك يفحص البرنامج محتويات السطر رمزا رمزًا ، فإذا ماكان أحد

الرموز للحروف المتحركة ( وبصورة أكثر تحديداً ... إذا ماكان الرمز عضوا فى فئة الرموز المحددة للحروف المتحركة ) يزداد عداد vowel بمقدار ١ . وبالمثل إذا ماكان أحد الرموز للحروف الساكنة ، فإن عداد الحروف الساكنة يزداد بمقدار .

وبعد ذلك تكتب نتيجة التحليل عن طريق الاتصال بالإجراء writeoutput ، ثم يتم إدخال سطر جديد داخل الكمبيوتر عن طريق إجراء readinput . وتعاد هذه الدورة حتى تدخل كلمة end فى بداية سطر جديد .

والاجراء المستخدم لقراءة سطر جديد من النص وتحديد إجمالى عدد الرموز ، يمكن أن يكتب على النحو التالى :

```
PROCEDURE readinput;
(* this procedure reads in a line of text *)
BEGIN
   writeln('Please enter a line of text below');
   count := 0;
   WHILE NOT eoln DO
      BEGIN
         count := count + 1;
         read(line[count])
      END;
   readln;
   charcount := count
END;
```

وهذا الإجراء مباشر ، ولايحتاج إلى أى مناقشة .

كما أنإجراء المخرجات writeoutput مباشر أيضا . وهذا الإجراء يكتب القيم النهائية للثلاثة عدادات con-scount , vowelcount , charcount . وعلى هذا ... يمكننا كتابة هذا الإجراء كما يلى :

```
PROCEDURE writeoutput;
(* this procedure writes out an analysis of a line of text *)
BEGIN
   writeln;
   writeln('Number of characters: ',charcount:2);
   writeln('Number of vowels    : ',vowelcount:2);
   writeln('Number of consonants: ',conscount:2);
   writeln
END;
```

وفيما يلى البرنامج كاملا :

```
PROGRAM vowelcount(input,output);
(* THIS PROGRAM COUNTS THE TOTAL NUMBER OF CHARACTERS,
   THE NUMBER OF VOWELS AND THE NUMBER OF CONSONANTS
   APPEARING IN A LINE OF TEXT *)
TYPE letters = SET OF char;
VAR vowels,consonants : letters;
    count,charcount,vowelcount,conscount : 0..80;
    line : PACKED ARRAY [1..80] OF char;

PROCEDURE readinput;
(* this procedure reads in a line of text *)
BEGIN
   writeln('Please enter a line of text below');
   count := 0;
   WHILE NOT eoln DO
      BEGIN
```

(تكملة البرنامج فى الصفحة التالية)

```
            count := count + 1;
            read(line[count])
         END;
      readln;
      charcount := count
   END;

   PROCEDURE writeoutput;
   (* this procedure writes out an analysis of a line of text *)
   BEGIN
      writeln;
      writeln('Number of characters: ',charcount:2);
      writeln('Number of vowels    : ',vowelcount:2);
      writeln('Number of consonants: ',conscount:2);
      writeln
   END;

   BEGIN   (* main action block *)
      vowels := ['A','E','I','O','U','a','e','i','o','u'];
      consonants := ['A'..'Z','a'..'z'] - vowels;
      readinput;
      WHILE NOT ((line[1] IN ['E','e']) AND
                 (line[2] IN ['N','n']) AND
                 (line[3] IN ['D','d'])) DO
         BEGIN
            vowelcount := 0;
            conscount := 0;
            FOR count := 1 TO 80 DO
               BEGIN
                  IF line[count] IN vowels
                     THEN vowelcount := vowelcount + 1;
                  IF line[count] IN consonants
                     THEN conscount := conscount + 1
               END;
            writeoutput;
            readinput
         END
   END.
```

اعتبر الآن مايحدث عندما ينفذ هذا البرنامج . افرض على سبيل المثال أننا أدخلنا السطرين التاليين :

```
Pascal is a structured programming language derived from ALGOL-60
end
```

ينتج عن تنفيذ البرنامج الحوار التالى (استجابات المستفيد موضوع تحتها خط) .

```
Please enter a line of text below
Pascal is a structured programming language derived from ALGOL-60

Number of characters: 65
Number of vowels    : 20
Number of consonants: 34

Please enter a line of text below
end
```

## أسئلة مراجعة : Review Questions

(١) ماهى الخواص الأساسية للفئة ؟ وكيف تختلف الفئة عن المنظومة ؟ وكيف تختلف الفئة عن السجل ؟

(٢) ماذا يعنى النوع الأساسى ؟ وكيف يعرف النوع الأساسى ؟

(٣) ماذا يعنى نوع الفئة ؟ وكيف يختلف نوع الفئة عن النوع الأساسى ؟ وكيف يعرف نوع الفئة ؟

(٤) أى نوع من أنواع البيانات يمكن أن يوجد داخل الفئة ؟

(٥) هل يمكن تعريف أنواع فئات مختلفة لها نفس النوع الأساسى ؟

(٦) ماهو الغرض من المتغير من نوع الفئة ؟ وماهو نوع عنصر البيانات الذى يمثله هذا المتغير ؟

(٧) لخص قواعد إعداد الفئة ؟

(٨) ماهو أقل عدد من العناصر يجب أن يوجد فى الفئة ؟

(٩) عند إعداد الفئة ، هل يجب أن تحدد العناصر الفردية للفئة بترتيب معين ؟

(١٠) ماهى القيود الموضوعة على استخدام متغيرات من نوع الفئة عند إعداد الفئة ؟

(١١) عند إعداد الفئة ، ماهى العناصر المتتالية فى الفئة التى يعبر عنها بالمدى الجزئى ؟

(١٢) عند إعداد الفئة ، ماذا يحدث إذا ماتم تحديد نفس عنصر الفئة أكثر من مرة واحدة ؟

(١٣) افرض أن مواصفات الفئة تحتوى على مدى جزئى واحد ، ومواصفات المدى الجزئى مكتوبة بترتيب خاطئ . كيف يفسر ذلك ؟ وضح إجابتك .

(١٤) لخص قواعد تحديد فئة لمتغير من نوع الفئة . ماهى القيود الموضوعة بالنسبة لتوافقية النوع ؟

(١٥) ماذا يعنى اتحاد فئتين ؟ كيف يمكن أداء هذه العملية فى البسكال ؟ وماهى القيود الموضوعة على العناصر التى تجرى عليها هذه العملية ؟

(١٦) ماذا يعنى تقاطع فئتين ؟ كيف يمكن أداء هذه العملية فى البسكال ؟ وماهى القيود الموضوعة على العناصر التى تجرى عليها هذه العملية ؟

(١٧) ماذا يعنى الفرق بين فئتين ؟ كيف يمكن أداء هذه العملية فى البسكال ؟ وماهى القيود الموضوعة على العناصر التى تجرى عليها هذه العملية ؟

(١٨) ماهى المؤثرات العلاقية التى يمكن استخدامها مع عناصر من نوع الفئة ؟ وماهى المؤثرات العلاقية المستخدمة فى ذلك ؟ وكيف يفسر كل منها ؟

(١٩) ماذا يعنى اختبار العضوية ؟ وكيف يؤدى هذا الاختبار فى البسكال ؟

(٢٠) مانوع العناصر التى يمكن استخدامها مع المؤثر العلاقى IF ؟ قارن ذلك باستخدام المؤثرات العلاقية الأخرى مع عناصر من نوع الفئة .

## مسائل محلولة : Solved Problems

(٢١) فيما يلى عدة تعريفات توضيحية لفئات وتوضيحات لمتغيرات من نوع الفئة .

```
(a) VAR vowels,consonants : SET OF char;
(b) VAR caps : SET OF 'A'..'Z';
(c) TYPE capitals = 'A'..'Z';
    VAR  caps : SET OF capitals;
(d) TYPE capitals = 'A'..'Z';
         uppers = SET OF capitals;
    VAR  caps : uppers;
(e) VAR movement : SET OF (north,south,east,west);
(f) TYPE compass = (north,south,east,west);
    VAR  movement : SET OF compass;
(g) TYPE compass = (north,south,east,west);
         direction = SET OF compass;
    VAR  nextmove,lastmove : direction;
```

(٢٢) حدد العناصر الموجودة فى كل فئة من الفئات التالية ، طبقا للتوضيحات التالية :

```
TYPE weekdays = (sun,mon,tue,wed,thu,fri,sat);
     days = SET OF weekdays;
```

```
(a) [mon..fri]                  (d) [fri..mon] + [wed]
(b) [mon..fri] + [wed]          (e) [tue] - [mon..sat]
(c) [mon..fri] - [wed]          (f) [sun..wed] * [tue..sat]
```

```
(a) [mon,tue,wed,thu,fri]       (d) [wed]
(b) [mon,tue,wed,thu,fri]       (e) [ ]
(c) [mon,tue,thu,fri]           (f) [tue,wed]
```

(٢٣) حدد الناتج من كل عبارة من عبارات التحديد التالية ، طبقا للتوضيحات التالية :

```
TYPE weekdays = (sun,mon,tue,wed,thu,fri,sat);
     days = SET OF weekdays;
VAR workdays,restdays : days;
    today : weekdays;
(a) workdays := [mon,wed..sat] + [sun];
(b) restdays := [tue,thu];
        .
        .
        .
    workdays := [mon..sat] - restdays;
```

```
(c) workdays := [ ];
    FOR today := mon TO thu DO
        workdays := workdays + [today];
    restdays := workdays * [thu..sat];
```

(*a*) workdays = [sun,mon,wed,thu,fri,sat]

(*b*) workdays = [mon,wed,fri,sat]

(*c*) workdays = [mon,tue,wed,thu]
restdays = [thu]

(٢٤) حدد قيمة كل تعبير من تعبيرات بوليان التالية طبقا للتوضيحات التالية :

```
TYPE weekdays = (sun,mon,tue,wed,thu,fri,sat);
     days = SET OF weekdays;
VAR workdays,restdays : days;
    today : weekdays;
```

```
(a) [sun,sat] = [sun..sat]
(b) [tues,fri] >= []
(c) [mon,wed,fri] <= [mon..fri]
(d) [fri,mon..wed] = [mon,wed,fri]
(e) workdays := [mon..fri];
    ..............................
    workdays <> [mon,tue,wed,thu,fri
(f) today := fri;
    restdays := [fri,sat,sun];
    ..........................
    [today] <= restdays
(g) mon IN [mon,wed,fri]
(h) sun IN [sun..sat]
(i) sat IN [mon,wed,fri]
(j) today := wed;
    ...................
    today IN [mon,wed,fri]
(k) today := sat;
    workdays := [mon..fri];
    ....................
    today IN workdays
```

(*a*) false (*e*) false (*i*) false
(*b*) true (*f*) true (*j*) true
(*c*) true (*g*) true (*k*) false
(*d*) false (*h*) true

## Supplementary Problems مشاكل متكاملة :

(٢٥) حدد العناصر الموجودة فى كل فئة من الفئات التالية طبقا للتوضيحات التالية :

```
TYPE notes = (do,re,mi,fa,sol,la,ti);
     song  = SET OF notes;
```

```
(a) [re..la]                  (e) [do..fa] - [la..re]
(b) [do..fa] + [re..la]       (f) [re,fa,sol] - [re..la]
(c) [do..fa] - [re..la]       (g) [re,fa,sol] * []
(d) [do..fa] * [re..la]       (h) [do,re,mi] + [re]
```

(٢٦) حدد الناتج من كل عبارة من عبارات التحديد التالية ، طبقا للتوضيحات التالية :

```
TYPE notes = (do,re,mi,fa,sol,la,ti);
     song  = SET OF notes;
VAR ballad,disco : song;
    tone : notes;
```

(*a*) `ballad := [do..fa] * [re,fa];`

(*b*) `disco := [do..fa] - [mi];`

(*c*)
```
tone := ti;
   .
   .
   .
ballad := [re..fa,la] + [tone];
```

(*d*)
```
disco := [do,mi,sol];
   .
   .
   .
ballad := disco + [re,ti];
```

(*e*)
```
disco := [];
FOR tone := la DOWNTO re DO
   disco := disco + [tone];
```

(٢٧) حدد قيمة كل تعبير من تعبيرات بوليان التالية ، طبقا للتوضيحات التالية :

```
TYPE coins = (penny,nickel,dime,quarter,half,dollar);
     change = SET OF coins;
VAR money : coins;
    wine,whiskey,song : change;
```

(*a*) `[penny,nickel] <= [penny,quarter]`

(*b*) `[nickel,dime,quarter] = [nickel..quarter]`

(*c*) `[penny..dollar] = [dollar..penny]`

(*d*) `[nickel..quarter] >= [dime,quarter]`

(*e*) `[half..nickel] <> []`

(*f*) `dollar IN [penny..dollar]`

(*g*)
```
money := dollar;
. . . . . . . . .
[money] <= [penny..dollar]
```

(*h*)
```
money := dollar;
. . . . . . . . .
money IN [penny..dollar]
```

(*i*)
```
wine := [penny,nickel,dime];
. . . . . . . . . . . . . .
dollar IN wine
```

(*j*)
```
whiskey := [penny,nickel,dime];
. . . . . . . . . . . . . . . . .
dime IN whiskey
```

(*k*)
```
wine := [penny,nickel,dime];
song := [dime..dollar];
. . . . . . . . . . . . .
dime IN wine*song
```

(*l*)
```
money := penny;
song := [dime..dollar];
. . . . . . . . . . . . .
money IN song
```

(٢٨) فيما يلى العديد من العبارات أو مجموعات العبارات التى تحتوى على فئات أو أعضاء فئات ، طبقا للتوضيحات المعطاه فى المشكلة السابقة . بعض هذه العبارات غير صحيحة . عرف كل الأخطاء .

(*a*)
```
money := dime;
   .
   .
   .
WHILE money IN [penny,nickel,dime] DO
   BEGIN
      .
      .
      .
   END;
```

(*b*)
```
money := dime;
   .
   .
   .
WHILE money <= [penny,nickel,dime] DO
   BEGIN
      .
      .
      .
   END;
```

(*c*)
```
song := [];
   .
   .
   .
FOR money := penny TO dollar DO
   song := song + money;
```

(*d*)
```
IF NOT (quarter IN wine) THEN
   wine := wine + [quarter];
```

(*e*)
```
IF [penny] <= song THEN song := song - [penny];
```

(*f*)
```
IF whiskey IN [penny..quarter] THEN
   whiskey := whiskey + [half];
```

## مشاكل برمجة : Programming Problems

(٢٩) عدل البرنامج الموجود فى مثال (١٢ - ١٦) بالطرق التالية :

أ - استخدام اختبار عضوية داخل البرنامج ( أى إحلال تعبيرات بوليان بتـعبيرات مكافـئة لاتستخدم المؤثر IN ) .

ب - تحديد العناصر غير الحرفية ( أى الأرقام وعلامات التنقيط وخلافه ) ، الموجودة بالإضافة الى الرموز الموجودة .

ج - كتابة القوائم الأربع التالية بعد تحليل كل سطر .

١ - الحروف الموجودة فى سطر معين من أسطر النص .

٢ – الرموز غير الحرفية الموجودة فى سطر معين من أسطر النص .

٣ – الحروف غير الموجودة فى النص .

٤ – الرموز غير الحروف غير الموجودة .

اختبر البرنامج مستخدما عدة أسطر مدخلات من إختيارك الشخصى .

(٣٠) وسع البرنامج الموجود فى مثال ( ١٢ – ١٩ ) ليحتوى على المعالم التالية :

أ – أحسب عدد عناصر كل الحروف المتحركة التى تظهر داخل السطر .

ب – تحديد عدد الكلمات الموجودة فى السطر ( ملاحظة عدد الفراغات الموجودة فى السطر )

ج – تحديد متوسط طول كل كلمة داخل السطر .

اختبر البرنامج ، مستخدما العديد من الأسطر التى تختارها بنفسك

(٣١) أكتب برنامجا متداخلا بلغة البسكال ، يحول التاريخ الذى تم إدخاله فى صورة mm - dd - yy (مثل - 22 - 4 69) إلى رقم صحيح يحدد عدد الأيام بعد أول يناير ١٩٦٩ ، مستخدما الطريقة المذكورة فى المشكلة رقم (٥٠ من الفصل السادس ) الجزء ( ر ) على أن يحتوى البرنامج على اختبارات للخطأ ، بحيث يمكن اكتشاف بيانات المدخلات الخاطئة ، وإنتاج رسالة خطأ مناسبة . استخدم تكوين بيانات من نوع الفئة فى جزء اختبار الخطأ .

(٣٢) أعد كتابة البرنامج المتداخل الخاص بلعبة tic - tac - toc الموجود فى المشكلة رقم ( ٥٠ من الفصل السابع ) ( الجزء i ) ، بحيث يشمل البرنامج جزءا للتأكد من الخطأ الخاص ببيانات المدخلات . يجب أن يختبر هذا الجزء النقل الخاطئ والاستجابات الخاطئة للفئات المدخلات . استخدم تكوين بيانات من نوع الفئة فى هذا الغرض .

(٣٣) أعد كتابة برنامج منتج pig latin المذكور فى المشكلة رقم ( ٥٠ من الفصل التاسع ) ، بحيث يستخدم تكوين بيانات من نوع الفئة عند اختبارات علامات التنقيط وأصوات الأحرف المزدوجة والحروف الكبيرة . هل استخدام الفئة هو أفضل طريقة لتنفيذ كل من هذه الاختبارات ؟

(٣٤) تصف المشكلة رقم ( ٥٢ من الفصل التاسع ) تطبيق برمجة يحسب فيه متوسط درجات الطلاب فى امتحانات البسكال ، وتحديد متوسط درجة الفصل . افرض الآن أنه تم تعريف خمسة مجالات عددية ، بحيث يقع المتوسط الشامل للفصل فى منتصف المجال الثالث . ويمكن أن يحدد حرف كمعدل لكل طالب ، وذلك طبقا للمدى الخاص الذى يقع فيه متوسط كل طالب ... وعلى هذا ... فإن أول مدى ( أعلى مدى ) يمثل A ، والثانى يمثل B وهكذا .

وإحدى الطرق لعمل المجالات العددية الفردية هى كما يلى : دع CA تمثل المتوسط الشامل للفصل . واحسب خارج القسمة .

$$Q = \frac{(100 - CA)}{3}$$

ثم احسب المدى كما يلى :

| | | | | |
|---|---|---|---|---|
| first range | (A): | (CA + 2Q) | to | 100 |
| second range | (B): | (CA + Q) | to | (CA + 2Q) |
| third range | (C): | (CA − Q) | to | (CA + Q) |
| fourth range | (D): | (CA − 2Q) | to | (CA − Q) |
| fifth range | (F): | below (CA−2Q) | | |

وعلى هذا إذا كان المتوسط الشامل للفصل ٧٠ ٪ ، فإن Q تساوى 10=3/ (70 - 100) ، وتصبح المجالات كما يلى :

A: 90 to 100
B: 80 to 90
C: 60 to 80
D: 50 to 60
F: below 50

اكتب برنامجا متداخلا بلغة البسكال ينفذ هذه العملية ، مفترضا أوزانا ( ترجيماتا ) متساوية لكل الامتحانات . أستخدم كل من تكوينات بيانات من نوع السجل ، ومن نوع الفئة فى البرنامج . اختبر البرنامج مستخدما البيانات المعطاه فى المشكلة رقم ( ٥٢ من الفصل التاسع ) .

(٣٥) فيما يلى طريقة للحصول على قائمة بالأعداد الأولية التى تقع داخل المدى الذى يتراوح من 2 إلى n .

أ - إنتاج قائمة مرتبة بالأرقام الصحيحة من 2 إلى n .

ب - أداء العمليات التالية لبعض الأرقام الخاصة الموجود فى القائمة .

١ - كتابة الرقم وإضافته إلى قائمة الأعداد الأولية .

٢ - نقل كل الأرقام المتتالية التى تقبل القسمة على i .

ج - إعادة الجزء ( b ) لكل قيمة متتالية من قيم i ، بدءا بقيمة i = 2 ، ومنتهيا بآخر رقم متبق .

عادة مايشار إلى هذه الطريقة بأنها منخل اراتو ستينز Sieve of Eratosthenes .

اكتب برنامجا بلغة البسكال يستخدم هذه الطريقة فى تحديد الأعداد الأولية الموجودة فى القائمة ، والتى تتراوح من 1 إلى n حيث n هى كمية يتم إدخالها للبرنامج .

استخدم تكوينات بيانات من نوع الفئة داخل البرنامج .

(٣٦) ترغب إحدى وكالات العمل فى حفظ قائمة باستخدام الكمبيوتر للوظائف المتاحة حاليا . وداخل هذه القائمة يتم وصف كل وظيفة براتبها الشهرى ، أو بأى صفات أخرى تميزها .

وعندما يتصل أحد الأشخاص الباحثين عن عمل بهذه الوكالة ، يتم إدخال معلومات وصفية عن هذا الشخص داخل الكمبيوتر لتقارن مع الخواص التى تميز الوظائف المتاحة ، ثم ينتج عند ذلك قائمة بالوظائف ذات الرواتب المرتفعة الجيدة ، والتى تناسب هذا الشخص .

وتستخدم الصفات التالية فى تمييز مهارات العمل المطلوب .

| Attribute | Job Skill |
|---|---|
| A | accounting |
| B | business |
| C | computer programming |
| D | dental technology |
| E | engineering/technical |
| F | food service |
| M | medical technology |
| P | personnel administration |
| R | receptionist |
| S | sales |
| T | typing/word processing |
| U | unskilled labor |

كما يكون مطلوبا صفات أخرى تحدد المستوى التعليمى المطلوب ، وهى :

| Attribute | Educational Level |
|---|---|
| 1 | high school |
| 2 | vocational |
| 3 | college |
| 4 | postgraduate |

اكتب برنامجا بلغة البسكال يمكنه أن يلبى احتياجات وكالة العمل هذه . استخدم تكوينات بيانات من نوع الفئة لعمل التوافق المطلوب بين الشخص والوظائف .

اختبر البرنامج مستخدما البيانات التالية :

Positions Available

| Position Number | Monthly Salary, $ | Required Job Skills | Education Level |
|---|---|---|---|
| 1 | 625 | U | 1 |
| 2 | 1350 | A,T | 3 |
| 3 | 900 | S | 3 |
| 4 | 2400 | E,B | 4 |
| 5 | 450 | F | 1 |
| 6 | 1100 | P,T | 1 |
| 7 | 1700 | D,T,R | 1 |
| 8 | 200 | M | 2 |

Potential Employees

| Client Number | Minimum Req'd Salary, $ | Job Skills | Education Level |
|---|---|---|---|
| 1 | 800 | R,T | 1 |
| 2 | 1200 | A,C,T | 3 |
| 3 | 2000 | E,B,C | 4 |
| 4 | 400 | U | 1 |
| 5 | 800 | S,P,T | 3 |

(٣٧) عدل البرنامج المكتوب فى المشكلة السابقة ، بحيث تخزن المعلومات التى تصف الوظائف المتاحة فى ملف بيانات . استخدم تكوينات بيانات من نوع السجل لتمثيل كل وظيفة من الوظائف المتاحة . أدخل فى البرنامج جزءا لإضافة سجلات ، ولحذف سجلات ، ولتغيير سجلات ، ولسرد كل السجلات .

(٣٨) ترغب إحدى الجامعات فى إنتاج إجراء تسجيل للطلاب آليًا ، حيث يستخدم كل قسم جهاز كمبيوتر صغيراً خاصاً به ، وقاعدة بيانات خاصة به لهذا الغرض . تحتوى قاعدة البيانات على قائمة بكل المقررات التى يجب أن يدرسها الطالب ليحصل على درجته الجامعية ، وتحتوى على المتطلبات السابقة أيضا . وسوف تخزن هذه القائمة بالطبع فى ملف بيانات .

ويجب أن تدرس المقررات بترتيب عددى ، فيما عدا أن الطلاب لايستطيعون التسجيل فى مقرر إلا إذا نجحوا فى المقررات الأخرى التى تعتبر متطلبات أساسية لهذا المقرر . ويصبح هذا الموقف معقدا بعض الشئ ، نظرا للحقيقة التى تقول إن الطلبة يرسبون فى مقرر ، ويخرجون من التسلسل على هذا . كما أن بعض الطلبة ينتقلون أيضا إلى برنامج دراسة ، وعلى هذا ... فلا يدرسون كل المتطلبات السابقة .

اكتب برنامجا بلغة البسكال يسمح لكل طالب فى أحد الأقسام أن يسجل نفسه فى ثلاثة مقررات جديدة ، كل فصل دراسى مختاراً من المقررات الموجودة فى ملف البيانات ( افرض أن كل المقررات الموجودة فى ملف البيانات تعرض كل فصل دراسى )

يجب اختيار المقرارات طبقا لتسلسلها العددى المناسب ، مع افتراض أن الطالب أتم كل المتطلبات السابقة للمقرر الذى يسجل له . استخدم تكوينات بيانات من نوع الفئة لأداء هذا الاختبار .

اختبر البرنامج مستخدما البيانات التالية :

( المتطلبات السابقة للمقرر موضحة بين قوسين بعد رقم المقرر )

Courses

| | | | |
|---|---|---|---|
| 101 | 201 (101) | 301 (204) | 401 (301) |
| 102 | 202 | 302 (102,202) | 402 (302) |
| 103 | 203 (106) | 303 | 403 |
| 104 (101) | 204 (201) | 304 (301) | 404 (304) |
| 105 (102) | 205 | 305 (302) | 405 (205) |
| 106 | 206 (102,203) | 306 | 406 (102,306) |

Student Records

| Name | Courses Taken |
|---|---|
| Smith | 101,103 |
| Brown | 101,102,103,104,105,202 |
| Richardson | 101,102,103,104,105,106,201,203,204 |
| Davis | 101,102,104,105,201,203,204,206,301,304 |
| Thomas | 101,102,103,104,105,106,201,202,203,204,<br>205,206,302,303,305,306,402,405 |

# الفصل الثالث عشر

# القوائم والمشيرات (١)

# Lists and Pointers

اهتممنا فى الفصول من الفصل التاسع وحتى الفصل الثانى عشر بأنواع مختلفة متعددة لتكوينات البيانات ، وهى : المنظومة ، والسجل ، والملف ، والفئة .

وبالرغم من أن كل نوع من هذه الأنواع للبيانات له خواصه الخاصة به ، فكلها تشترك فى خواص معينة . فمثلا أقصى عدد لمكونات منظومة أو سجل أو فئة يحدد فى توضيح البيانات ، ويظل ثابتا طوال فترة تنفيذ البرنامج . أكثر من هذا ... ترتب المكونات الفردية داخل أى نوع من أنواع تكوينات البيانات فى ترتيب ثابت بالنسبة لبعضها . ( وهذا صحيح بالنسبة لكل تكوينات البيانات ، بما فيها الملفات ) . وتميل هذه الخواص إلى تقييد المنفعة من أنواع تكوينات البيانات فى بعض التطبيقات .

وبصفة خاصة تتطلب بعض التطبيقات استخدام قوائم ، تكون مكوناتها متصلة مع بعضها بواسطة مشيرات pointers . وعلى هذا ... يشار إلى مثل هذه القوائم بانها قوائم متصلة linked lists . وتستخدم القوائم المتصلة على سبيل المثال مع المترجمات ، ونظم التشغيل ، ونظم إدارة قواعد البيانات . وعلى هذا ... فإننا نعيد انتباهنا الآن إلى استخدام القوائم المتصلة فى البسكال .

## ١ – أساسيات    1. PRELIMINARIES

الفكرة الأساسية للقائمة المتصلة هى أن كل عنصر فردى داخل القائمة يحتوى على مشير يحدد المكان الذى يمكن أن يوجد فيه العنصر التالى له . وعلى هذا ... فإن الترتيب النسبى للمكونات يمكن تغييره ببساطة ، وذلك بتغيير المشـير . بالإضافة إلى ذلك ... يمكن إضافة المكونات الفردية بسهولة إلى القائمة ، أو حذفها منها ، وذلك بتغيير المشير أيضا . وعلى هذا ... فإن القائمة المتصلة لاتقتصر على حد أقصى لعدد مكوناتها . ومثل هذه القائمة يمكن توسيعها أو تضييقها بالنسبة لحجمها ، والبرنامج الذى يستخدمها .

### مثـال (١٣-١)

يوضح الشكل ( a ) ١٣- ١ قائمة متصلة تحتوى على ثلاثة مكونات . كل مكون من هذه المكونات الثلاثة يحتوى على عنصر بيانات : قيمة متعددة ( لون ) ، ومشير يشير إلى العنصر التالى فى القائمة .

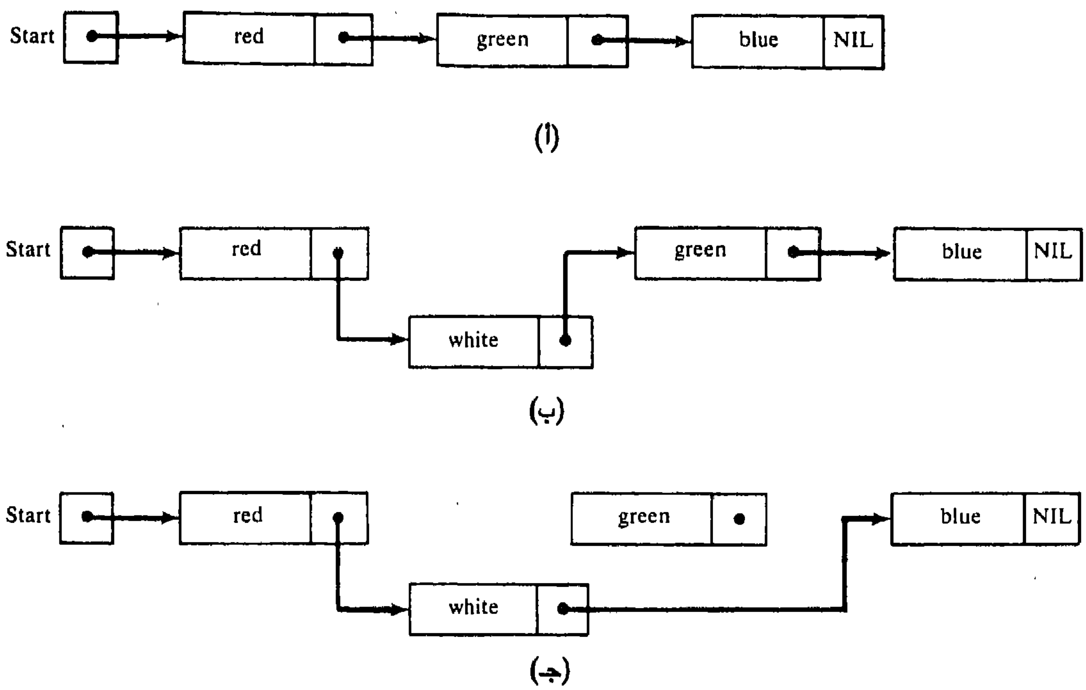
وعلى هذا ... فأول مكون له القيمة red ، والثانى green ، والثالث blue . وتتحدد بداية القائمة بمشير منفصل يسمى start ، كما تتحدد نهايتها بالقيمة NIL ( وسوف يذكر المزيد عن ذلك فيما بعد ) .

دعنا نضيف الآن مكونا جديدا قيمته white بين red , green . لعمل ذلك ... فإننا نغير المشير كما هو موضح فى شكل ( b ) ١٣- ١

---

(١) استخدم أسم مشير للدلالة على الكلمة الإنجليزية pointer ، وذلك لأنه سبق استخدام كلمة مؤشر للدلالة على الكلمة الإنجليزية parameter.

فإذا ماأردنا أن نحذف المكون الذى قيمته green فإننا نغير المشير ببساطة الذى يصاحب المكون الثانى كما هو موضح فى شكل ( c )١٣- ١ .

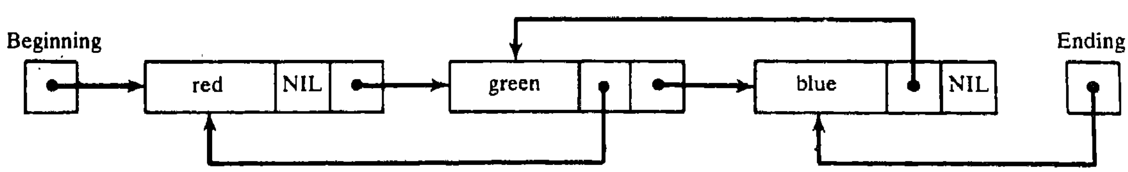



شكل (١٣- ١ )

وهناك أنواع عديدة مختلفة من التكوينات المتصلة تشمل القوائم الخطية linear المتصلة ( حيث تتصل كل مكوناتها معا بطريقة متسلسلة ) . والقوائم المتصلة بعدة مشيرات ( لتسمح بالحركة للأمام وللخلف داخل القائمة ) . والقوائم الدائرية circular ( وهى قوائم خطية ليس لها بداية ونهاية ) . والأشجار trees ( والتى ترتب فيها المكونات ترتيبا هرميا ) . وقد رأينا بالفعل توضيحا لاتصال خطى فى مثال (١٣- ١ ) . وفى المثال التالى يوجد بعض أنواع الاتصالات الأخرى .

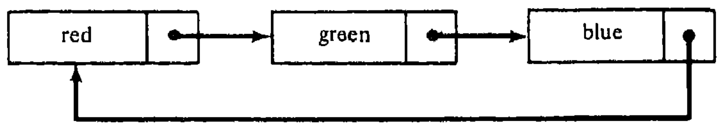
### مثال (١٣-٢)

نرى فى الشكل (١٣- ٢ ) قائمة اتصال خطية تشبه القائمة الموجودة فى شكل ( a )١٣- ١ . والآن نرى على أية حال أن هناك مشيرين مصاحبين لكل مكون من المكونات ، وهما مشير أمامى ومشير خلفى . ويسمح هذان المشيران بالحركة داخل القائمة فى كل من الاتجاهين ، أى من بدايتها وحتى نهايتها ، ومن نهايتها وحتى بدايتها .




شكل (١٣- ٢ )

اعتبر الآن شكل١٣– ٣ . هذه القائمة تشبه القائمة الموجودة فى شكل ( a )١٣– ٣ ، فيما عدا أن آخر عنصر بيانات ( blue ) يشير إلى أول عنصر بيانات ( red ) . وعلى هذا ... فهذه القائمة لايوجد لها بداية ونهاية . ويشار إلى مثل هذه القائمة بأنها قائمة دائرية .

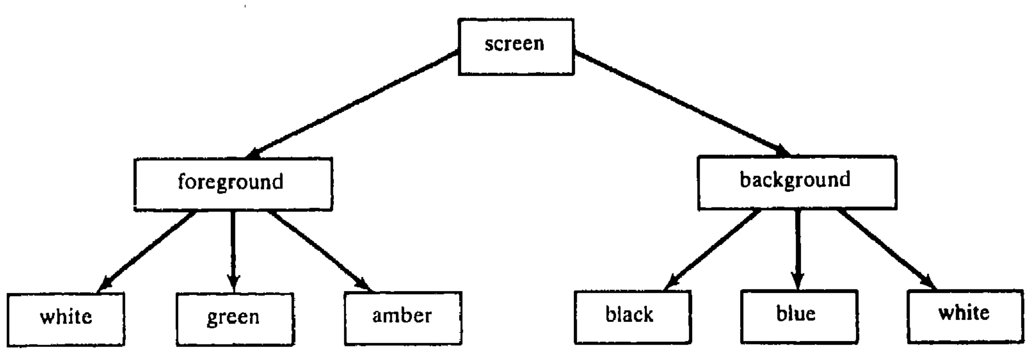



شكل (١٣– ٣ )

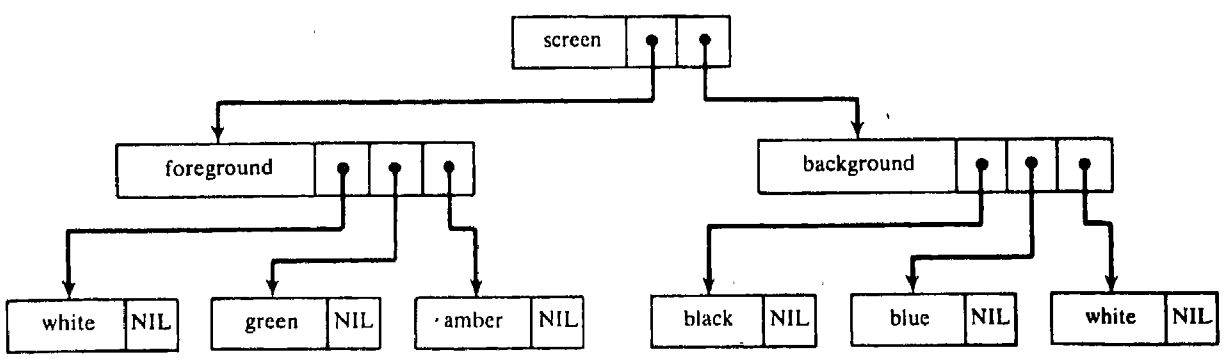
نرى أخيرًا فى شكل ( a )١٣– ٤ مثالا لشجرة . وتحتوى الشجرة على نقاط وأفرع مرتبة بطريقة هرمية ت، تحدد التكوين الهرمى المناظر للبيانات ( والشجرة الثنائية binary tree هى شجرة تكون كل نقطة فيها لها أكـثر مـن فرع واحد ) .

وفى شكل ( a )١٣– ٤ نقطة الجذر لها القيمة screen ، والأفرع المصاحبة تقود إلى النقاط التى قيمها هى foreground , background على التوالى . وبالمثل فالأفرع التى تصاحب foreground تعود إلى النقاط التى قيمها هى white , green , amber . والأفرع التى تصاحب background تعود إلى النقاط التى قيمها هى white , blue , black .

ويوضح شكل ( b )١٣– ٤ الطريقة التى تستخدم بها المشيرات فى أعداد الشجرة .




(أ)




(ب)

( شكل١٣– ٤ )

ولايحتوى البسكال على كل مكون متصل كنوع بيانات مستقل . بدلا من ذلك يستخدم نوع فردى للبيانات ، هو المشير الذى يسمح بإعداد أنواع مختلفة من التكوينات المتصلة ، طبقا لفئة محددة من القواعد .

## ٢ - تعريفات النوع : 2. TYPE DEFINITIONS

فى التطبيقات التى تحتوى على قوائم متصلة يجب أن يستخدم نوعان مختلفان من المتغيرات : متغيرات مشيرة pointer variables ( وهى متغيرات تشير قيمها إلى متغيرات أخرى ) ، ومتغيرات إشارة referenced variables ( وهى متغيرات " تشير إلى " ) . ويجب إعطاء اهتمام كبير لتعريف النوع الذى يصاحب كل نوع من أنواع المتغيرات . وهذا صحيح بصفة خاصة بالنسبة للنوع المشير ، pointer type أى تعريف النوع الذى يستخدم لتوضيح متغيرات من نوع المشير .

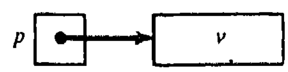
وبصفة عامة ... تعريف النوع المشير يكتب على النحو التالى :

```
TYPE pointer type = ↑type identifier
```

حيث يشير identifier بدون السهم إلى النوع المناظر لمتغير الإشارة . ( يظهر تعريف النوع هذا فى برنامج فيما بعد وذلك بعد تعريف نوع المشير ) .

والفكرة الأساسية من وراء تعريف النوع المشير هى مايلى :

تشير قيمة المتغير من نوع المشير إلى بعض متغيرات أخرى تم تحديد نوعها بواسطة نوع المعرف . وهذه الفكرة موضحة فى شكل ١٣- ٥ ، حيث نرى متغيراً من نوع المشير p تشير قيمته إلى متغير إشارة مناظر . وقيمة متغير الإشارة هذا يمثلها س . ومتغير الإشارة يكون من النوع المحدد بواسطة نوع المعرف identifier .




شكل (١٣- ٥ )

وعادة مايعرف متغير الإشارة كسجل تحتوى حقوله على مكونات داخل القائمة المتصلة . وعادة مايكون آخر حقل عبارة عن عنصر بيانات من النوع المشير ، يحدد موقع المكون التالى . ويمثل هذا الحقل الاتصال بين المكون الحالى والمكون التالى له .

### مثـال (١٣-٣)

اعتبر تعريفات النوع التالية :

```
TYPE primary = (red,green,blue);
     pointer = ↑hue;
     hue = RECORD
             color : primary;
             nextcolor : pointer
          END;
```

يحدد السطر الأول أن blue , green , red هى عناصر بيانات من النوع primary . ويعرف pointer فى السطر الثانى بأنه من نوع مشير يصاحب متغير إشارة من نوع hue . ( لاحظ أن متغير الإشارة هذا يستخدم لتمثيل مكون من مكونات قائمة الاتصال ) . أخيرا نعرف hue بأنه سجل يحتوى على حقلين ، هما color من نوع primary و nextcolor وهو مشير للمكون الثانى فى القائمة المتصلة .

لاحظ أن متغير الإشارة hue لم يعرف إلا بعد ظهوره فى تعريف المشير . وهذا يختلف عن تعريفات الأنواع الأخرى فى البسكال ، والتى يجب أن يعرف بها معرف قبل أن يكون ظهورها ممكنا فى أى تعريف نوع آخر .

وحيث إنه يمكن تعريف النوع المشير بالنسبة لنوع متغير الاشارة المناظر فقط ، فإننا نقول إن نوع المشير مرتبط bound بنوع متغير الإشارة الذى يشير إليه .

## ٣ – توضيحات المتغيرات : 3. VARIABLE DECLARATIONS

توضح المتغيرات من نوع المشير بالطريقة التقليدية ، أى توضح كما يلى :

```
VAR pointer name : type
```

حيث يشير pointer name إلى متغير من نوع المشير ، و type هو نوع المشير . وينتج عن هذا التوضيح متغير ساكن static أو استاتيكى ( تقليدى ) تشير قيمته إلى متغير إشارة ( أى إلى مكون داخل قائمة متصلة ) .

وعلى أية حال ... لاتوضح متغيرات الإشارة referenced varibles بالطريقة التقليدية ، حيث إن هذه المتغيرات يتم تحديدها وإلغاؤها بطريقة ديناميكية ، أى مع تنفيذ البرنامج . وعلى هذا ... يجب توضيح متغيرات الإشارة داخل مشيرات البرنامج التى تحتوى على عبارات إجرائية . ومثل هذه التوضيحات تتم بطريقة مختلفة عن توضيحات المتغيرات التى سبق التعرض لها من قبل .

ولعمل متغير إشارة ، فإننا نستخدم الإجراء القياسى new بكتابة مايلى :

```
new(pointer name)
```

ينتج عن هذه العبارة متغير جديد ( وهو متغير إشارة ) يعرف نوعه فى تعريفات النوع الرسمى . والمشير الذى يظهر إسمه فى عبارة new يشير تلقائيا إلى هذا المتغير الجديد . ويكون لمتغير الإشارة نفس الاسم مثل المتغير المشـير . وعلى أية حال ... يكتب متغير الإشارة مع وجود سهم رأسى بعد اسمه ، بحيث يمكن تمييزه عن المتغير المشير المناظر .

## مثال (١٣–٤)

اعتبر التخطيط الهيكلى التالى لأحد برامج البسكال .

```
PROGRAM sample(input,output);
TYPE primary = (red,green,blue);
     pointer = ↑hue;
     hue = RECORD
             color : primary;
             nextcolor : pointer
           END;
VAR foreground,background : pointer;
```

(تكملة البرنامج فى الصفحة التالية)

```
BEGIN   (* main action block *)
   .
   .
   new(foreground);
   .
   .
END.
```

تتكرر تعريفات النوع من مثال ١٣- ٣ . ويلى تعريفات النوع هذه توضيح أن المتغيرين background , foreground من النوع المشير .

لاحظ أن هذه توضيحات تقليدية للمتغيرات . وعلى هذا يكون كل من background , foreground متغيرين استاتيكيين يمكن تحديد قيمهما بالطريقة التقليدية .

ومن ناحية أخرى ... ينتج عن عبارة new الموجودة داخل المجموعة الاجرائية الرئيسية متغير إشارة من نوع hue يسمى foreground . وأكثر من ذلك .. فإن المتغير المشير المناظر foreground سوف يشير تلقائيا إلى foreground .

لاحظ أن متغير الإشارة foreground هو متغير من نوع السجل ، مكون من حقلين . وعلى هذا ... يمثل foreground , color قيمة متعددة من نوع primary ، ويمثل foreground , nextcolor مشيرا يمكن تحديده لمتغير الإشارة التالى :

## ٤ – العمليات مع المتغيرات المشيرة ومتغيرات الإشارة

## 4. OPERATIONS WITH POINTER VARIABLES AND REFERENCED VARIABLES

هناك نوعان من العمليات يمكن إجراؤها مع المتغيرات المشيرة ومتغيرات الإشارة ، وهما التحديد Assignment والمقارنة Comparison . ويجب أن يجرى كل من نوعى العمليات مع عناصر متشابهه ، أى يجب أن تحدد المشيرات أخرى ، أو تقارن المشيرات بمشيرات أخرى ، وتقارن متغيرات الإشارة بمتغيرات إشارة أخرى .

دعنا نعتبر أولا عملية التحديد . افرض أن كلاً من p2 , p1 متغير مشير من نفس النوع . يمكننا على ذلك تحديد القيمة p1 للمتغير p2 بكتابة مايلى :

```
p2 := p1
```

وبالمثل افرض أن كلاً من p2 , p1 متغير إشارة يحتوى على حقل من نوع المشير يسمى next . يمكننا على ذلك كتابة مايلى

( أنظر الصفحة التالية ) .

```
p2 := p1↑.next

p2↑.next := p1

p2↑.next := p1↑.next
```

وهكذا ، وفى كل الحالات نحدد قيمة أحد المتغيرات المشيرة لمتغير مشير آخر ( بافتراض أن متغيرات الإشارة يتم إنتاجها فى كل حالة بالطبع ) .

ويمكن أيضا أن يشير المتغير إلى لاشئ وذلك بتحديد القيمة الخاصة NIL لهذا المتغير ( لاحظ أن NIL هى كلمة محجوزة من كلمات البسكال ) . وعلى هذا ... فإذا لم نرد أن يشير p1 إلى أى شئ ، فإننا نكتب مايلى :

```
p1 := NIL
```

وسوف نرى استخداما لهذه السمة فيما بعد فى هذا الفصل .

لاحظ أننا رأينا الآن طريقتين مختلفتين لتحديد قيم لمتغيرات مشيرة . إحدى الطريقتين هى استخدام عبارة تحديد تقليدية كما سبق ذكره . والطريقة الأخرى هى استخدام الإجراء القياسى new كما سبق ذكره فى القسم السابق . ( تذكر أن القيمة تتحدد تلقائيا لمتغير مشير عند إعداد متغير إشارة مناظر عن طريق عبارة new ) .

يمكن أيضا تحديد متغيرات إشارة إلى متغيرات إشارة أخرى ، على أن تكون جميعها من نفس النوع . وعلى ذلك ... يمكننا أن نكتب مايلى :

```
p2↑ := p1↑
```

ويتسبب ذلك فى تحديد قيمة كل حقل موجود داخل p1 للحقل المناظر له داخل p2 .

ويمكن اجراء مثل هذا التحديد مع حقول فردية لمتغيرات إشارة ، أو مع محتويات متغيرات إشارة .

**مثـال (١٣-٥)**

اعتبر التوضيحات التالية ، والمكررة من المثال السابق .

```
TYPE primary = (red,green,blue);
     pointer = ↑hue;
     hue = RECORD
              color : primary;
              nextcolor : pointer
           END;
VAR foreground,background : pointer;
```

اذا ما أعد متغيرا الإشارة foreground↑ , background↑ ( عن طريق عبارة new ) . فكل مايلى يكون على ذلك تحديدات مشيرة صحيحة .

```
foreground := background
background := NIL
foreground := background↑.nextcolor
foreground↑.nextcolor := background
foreground↑.nextcolor := background↑.nextcolor
```

كما يمكننا أن نحدد أيضا بعض أو كل قيم background↑ , foreground↑ . وعلى هذا ... مكننا أن نكتب مايلى :

```
background↑ := foreground↑
background↑.color := foreground↑.color
```

وأخيرا يمكننا أن نحدد قيمة متعددة لحقل من نفس النوع فى أحد متغيرات الاشارة . فمثلا :

```
foreground↑.color := green
```

ويجب أن يفهم القارئ فهما كاملا الفرق بين التحديد الذى يشمل متغيرات مشيرة ، والتحديد الذى يشمل متغيرات إشارة . ولتوضيح هذه النقطة ، افرض أن المشيرين p2 , p1 يشيران إلى متغيرى إشارة مناظرين ، كما هو موضح فى شكل ( a )١٣- ٦

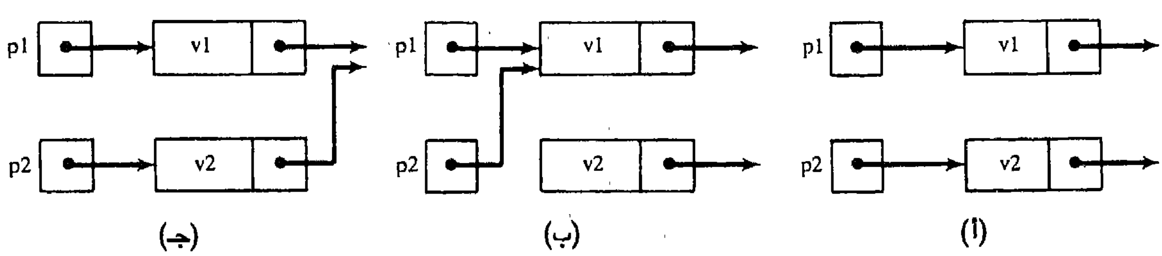
وعلى هذا ... فإن التحديد التالى :

```
p2 := p1
```

يتسبب فى أن يشير كل من المشيرين إلى متغير الإشارة p1↑ ، كما هو موضح فى شكل ( b )١٣- ٦ . ومن ناحية أخرى ... فإن التحديد :

```
p2↑ := p1↑
```

يتسبب فى تحديد قيم متغير الإشارة p1↑ لمتغير الإشارة p2↑ . وهذا موضح فى شكل ( c )١٣- ٦ .




شكل١٣- ٦

ويمكن مقارنة المتغيرات المشيرة باستخدام مؤثرى علاقيين ، هما = ، <> . وتسمح هذه السمة لتحديد ما إذا كان متغيران مشيران لهما نفس القيمة أو لا ، أو إذا ماكان متغير مشير تحددت له القيمة NIL .

## مثـال (١٣-٦)

اعتبر مرة أخرى التوضيحات المعطاة فى الأمثلة ٤ – ١٣ و ٥ – ١٣ ، أى اعتبر مايلى :

```
TYPE primary = (red,green,blue);
     pointer = ↑hue;
     hue = RECORD
              color : primary;
              nextcolor : pointer
           END;
VAR foreground,background : pointer;
```

فيما يلى عدة عبارات تحتوى على مقارنات بين متغيرات مشيرة background , foreground . ( وتميل هذه الأمثلة لتوضيح التكوين فقط ، وليس لها أى معنى منطقى خاص ) .

```
IF foreground = background THEN foreground := NIL

WHILE foreground <> background DO

REPEAT
   .
   .
   .
UNTIL background = foreground↑.nextcolor

IF background = NIL THEN foreground↑.color := red
```

ويمكن أيضا مقارنة متغيرات الإشارة ، على أن تكون من نفس النوع . ويمكن أن تشمل مثل هذه المقارنات محتويات متغيرات أو حقول فردية . فإذا ماقورنت حقول فردية ، فيجب أن تكون من نفس النوع أيضا .

## مثـال (١٣-٧)

فيما يلى بعض أمثلة لمقارنة متغيرات إشارة أو حقول فردية داخل متغيرات إشارة . وتستخدم مرة أخرى التوضيحات المقدمة فى الأمثلة السابقة ، أى :

```
TYPE primary = (red,green,blue);
     pointer = ↑hue;
     hue = RECORD
              color : primary;
              nextcolor : pointer
           END;
VAR foreground,background : pointer;
```

وفيما يلى بعض المقارنات الصحيحة .

```
IF foreground↑ = background↑ THEN foreground := NIL

IF foreground↑.color <> blue THEN foreground↑.color := blue

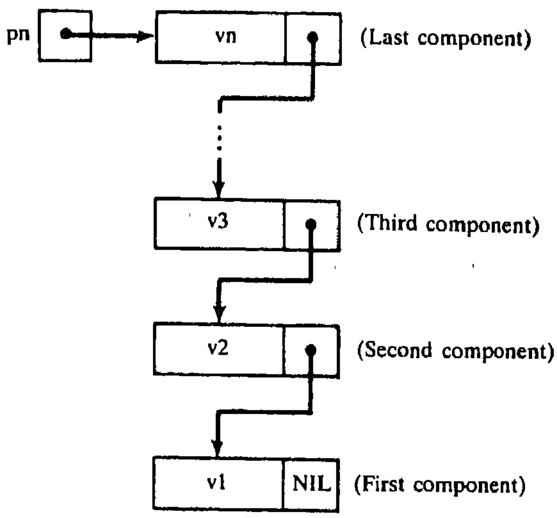
WHILE background↑.color <> foreground↑.color DO
```

## ٥ – إنتاج وإلغاء متغيرات ديناميكية

## 5. CREATING AND DESTROYING DYNAMIC VARIABLES

نعيد انتباهنا الآن إلى بعض المواقف التقليدية للبرمجة ، والتي تتطلب متغيرات مشيرة وما يناظرها من متغيرات إشارة . سوف نرى أن استخدام المؤشرات المشيرة يسمح بإنتاج وإلغاء متغيرات إشارة ديناميكيا ، أي مع تنفيذ البرنامج .

ولكي نوضح طريقة إنتاج وحذف متغيرات إشارة ديناميكيا ، فإننا نعمل مع النوع المعتاد من القائمة المتصلة ، التي يشير كل مكون من مكوناتها إلى المكون السابق له . وعلي هذا ... يشير المكون الثاني إلى المكون الأول ، ويشير الثالث إلى الثاني . وهكذا كما هو موضح في شكل١٣– ٧ . ويسمى هذا المخطط « الأخير في دخوله هو الأول في خروجه » last-in , first-out (LIFO) ، حيث إن آخر مكون يتم إدخاله في القائمة ، يكون أول مكون يغادرها .




شكل١٣– ٧

ويمكن إنتاج القائمة المتصلة بقراءة كل مكون جديد داخلها ، ثم توصيل هذه المكون بالمكون السابق له .

افرض على سبيل المثال أن item هو متغير من النوع المشير ، وأنه يناظر متغير اشارة ↑ item يحتوى على حقلين ، هما حقل سلسلة يسمى item↑ - name ، وحقل مشير اسمه item↑ , next . ويقدم المشير اتصالا للمكون الذى سبق إدخاله ، والذى يتحدد موقعه بواسطة متغير مشير pointer .

ولكى يتم إنتاج أول مكون ، فإننا ننتج متغيرًا ديناميكيًا ( item ) يقرأ قيمة item↑ , name فى القائمة ، ويضع قيمة NIL للمتغير item↑ , next . ويمثل هذا المكون الموجود فى قاعدة القائمة ، ثم يعد المتغير المشير ليشير إلى هذا المكون ( بحيث يمكن تعريف المكون السابق عندما يتم إدخال المكون الثانى ) .

وعلى هذا ... ينتج المكون الأول بكتابة مايلى :

```
new(item);
readname(item↑.name);
item↑.next := NIL;
pointer := item;
```

حيث read name هو إجراء يقرأ عناصر item↑ . name رمزا رمزا .

وتعاد هذه الخطوات لكل مكون تالٍ ، فيما عدا أن item↑ . next يعد ليشير إلى المكون السابق ، وذلك بتحديده للقيمة الحالية لـ pointer . وعلى هذا :

```
new(item);
readname(item↑.name);
item↑.next := pointer;
pointer := item;
```

يجب وضع مجموعة العبارات هذه داخل إحدى أنواع الدورات ( مثل مكون WHILE - DO ) ، بحيث يمكن تكراره لكل مكون تالٍ في القائمة .

ويمكن عرض القائمة ، وذلك بحركتها من القمة إلى القاعدة ( آخر مكون أولا ) . وفي البداية يحدد pointer موقعًا آخر ( أحدث ) مكونًا . وعلى هذا ... يمكننا أن نعرض محتويات القائمة ببساطة ، وذلك بتكرار الثلاث عبارات التالية لكل مكون من مكونات القائمة .

```
item := pointer;
writeln(item↑.name);
pointer := item↑.next;
```

ويجب أن توضع هذه العبارات داخل دورة ، يستمر تنفيذها حتى تصبح قيمة pointer هي NIL .

## مثـال (١٣-٨)

*إنتاج قائمة منفصلة* : فيما يلي برنامجًا كاملا بلغة البسكال ، ينتج ويعرض قائمة متصلة طبقا لمبدأ آخر عنصر يتم إدخاله يخرج أولا LIFO ، وذلك باستخدام الطريقة التي سبق ذكرها .

```
PROGRAM makelist(input,output);

(* THIS PROGRAM USES POINTERS TO CREATE A LINKED LIST OF NAMES *)

TYPE link = ↑personal;
     personal = RECORD
                   name : PACKED ARRAY [1..40] OF char;
                   next : link
                END;
VAR item,pointer : link;

PROCEDURE readname(VAR newname : link);
(* this procedure reads a name into the computer *)
VAR count : 0..40;
BEGIN
   FOR count := 1 TO 40 DO newname↑.name[count] := ' ';
   write('New name: ');
   count := 0;
   WHILE NOT eoln DO
      BEGIN
         count := count + 1;
         read(newname↑.name[count])
      END;
```

(تكملة البرنامج في الصفحة التالية)

```
      readln
   END;

   BEGIN   (* main action block *)

      BEGIN   (* create the list *)
         new(item);
         readname(item);
         item↑.next := NIL;
         pointer := item;
         WHILE NOT ((item↑.name[1] IN ['E','e'])
                AND (item↑.name[2] IN ['N','n'])
                AND (item↑.name[3] IN ['D','d'])) DO
                   BEGIN
                      new(item);
                      readname(item);
                      item↑.next := pointer;
                      pointer := item
                   END;
         pointer := item↑.next
      END;   (* create list *)

      BEGIN   (* display the list *)
         writeln;
         WHILE pointer <> NIL DO
            BEGIN
               item := pointer;
               writeln(item↑.name);
               pointer := item↑.next
            END;
      END   (* display list *)
   END.
```

يحتوى هذا البرنامج على إجراء اسمه readname يستخدم فى قراءة اسم يحتوى على عدد من الخانات ، يصل إلى ٤٠ خانة . ويبدأ الإجراء بتحديد مواقع فارغة لكل خانات الاسم . وبعد ذلك يتم إدخال الاسم الفعلى داخل الكمبيوتر رمزا رمزا ، حتى تكتشف نهاية السطر ( eoln ) والتى تتسبب فى إرجاع العربة .

لاحظ أن readname يستخدم مؤشرًا متغيرًا اسمه newname من نوع link . وعلى هذا ... فإن المؤشر عبارة عن متغير من نوع المشير . والقيمة التى تعود بواسطة هذا الإجراء ليست مشيرًا على أية حال ، لكنها إحدى العناصر الموجودة داخل متغير الإشارة المناظر ( المسمى newname↑ . name ) . وعلى ذلك ... فإننا نرى أن متغير الإشارة يمكن استغلاله داخل إجراء عن طريق نقل المتغير من النوع المشير المصاحب له إلى ( أو من ) الإجراء .

وتحتوى المجموعة الإجرائية الأساسية على عبارتين مركبتين ، إحداهما تنتج القائمة المتصلة ، والأخرى تعرضها على التوالى . والمنطق المستخدم داخل كل من هاتين العبارتين مباشر ، وذلك فى ضوء المناقشة التى سبقت هذا المثال . لاحظ على أية حال دورة WHILE - DO الموجودة فى أول عبارة . تتسبب هذه الدورة فى تكرار إدخال الأسماء الجديدة ، حتى يتم إدخال الكلمة end كاسم جديد ( سواء أكانت مكتوبة بحروف كبيرة أم حروف صغيرة ) . لاحظ أيضا أن المتغير من نوع المشير item يمرر إلى الإجراء readname .

افرض الآن أن البرنامج يستخدم لإنتاج قائمة متصلة تحتوى على الأسماء التالية :

Sharon, Gail, Susan, Marla, Marc, Amy, Megan

فيما يلى حوار المدخلات والمدخلات ، مع وضع خط تحت استجابة المستفيد .

```
New name: Sharon
New name: Gail
New name: Susan
New name: Marla
New name: Marc
New name: Amy
New name: Megan
New name: end


Megan
Amy
Marc
Marla
Susan
Gail
Sharon
```

لاحظ أن الأسماء مطبوعة فى ترتيب عكسى ، أى أن آخر اسم تم إدخاله طبع أولا .

ويمكن إدخال أى عنصر إضافى داخل بعض المواقع المحددة داخل القائمة المتصلة التى تم إعدادها . ولعمل ذلك يجب علينا أن ندخل العنصر الجديد ، محددين موقعه داخل القائمة ، ثم نضبط بعض المشيرات .

افرض – بصفة خاصة – أن item يمثل العنصر المراد إضافته ، وأن pointer يحدد العنصر الموجود خلف العنصر الجديد . تسمح العبارتان التاليتان بإدخال العنصر الجديد ( مفترضا أنه item↑ . name مرة أخرى ) .

```
new(item);
readname(item↑.name);
```

( يمكن تحديد ) الموقع المحدد للعنصر الجديد بكتابة مايلى :

```
readname(itemname);
pointer := last;
WHILE (pointer↑.name <> itemname) AND
      (pointer↑.next <> NIL) DO pointer := pointer↑.next;
```

حيث يكون item name متغير سلسلة يمثل العنصر التالى للعنصر الجديد ، ويكون last مشيرا لآخر عنصر داخل القائمة .

أخيرا يحدث تضبيط المشيرات كما يلى ، وذلك إذا ماكان يراد إدخال عنصر جديد فى نهاية القائمة .

```
item↑.next := last;
last := item;
```

وإلا فيكتب :

```
item↑.next := pointer↑.next;
pointer↑.next := item;
```

فى كل حالة من هذه الحالات يوجه جزء المشير للعنصر الجديد إلى العنصر الذى يسبقه ، والعنصر ( القديم ) الذى يلى العنصر الذى تمت إضافته يتم تضبيطه .

وحذف أحد العناصر يتم بطريقة تشبه إضافة عنصر جديد . فيتم الحذف الفعلى باستخدام عبارة dispose ، أى باستخدام .

```
dispose(pointer name)
```

وهذه العبارة متماثلة مع عبارة new . وعلى هذا ... فإن dispose هو اجراء قياسى يقبل متغيراً من النوع المشير كمؤشر . ويتسبب استخدام هذا الإجراء فى إلغاء متغير الإشارة المصاحب للمشير المعطى . وأى إشارة تالية لهذا المتـغير ( عن طريق مشير آخر ) تكون عند ذلك غير معرفة .

والطريقة الشاملة هى كما يلى :

نحدد أولا العنصر المراد حذفه ، ثم نعيد ضبط المشيرات ، وأخيرا نجرى dispose على العنصر المراد إزالته . وعلى هذا ... يكتب مايلى :

```
readname(itemname);
IF last↑.name = itemname
   THEN BEGIN   (* delete last item *)
           item := last;
           last := item↑.next;
           dispose(item)
        END
   ELSE BEGIN   (* find item, then delete *)
           pointer := last;
           item := last↑.next;
           WHILE (item↑.name <> itemname) AND
                 (item↑.next <> NIL) DO
              BEGIN
                 pointer := item;
                 item := item↑.next
              END;
           IF item↑.name = itemname THEN
              BEGIN
                 pointer↑.next := item↑.next;
                 dispose(item)
              END
        END;
```

يمكن لهذه العبارة أن تحقق موقفين مختلفين : حذف آخر عنصر ، وحذف بعض العناصر الأخرى . فإذا ماأريد حذف آخر عنصر ، فيتم ضبط المشير إلى نهاية القائمة ( last ) ، بحيث يشير إلى العنصر التالى فى القائمة ،

أى أن :

```
last := item↑.next;
```

أما إذا ماأريد حذف بعض العناصر الأخرى ، فيجعل العنصر التالى للعنصر الذى يحذف مشيرا إلى العنصر التالى له فى القائمة ، أى أن :

```
pointer↑.next := item↑.next;
```

## مثـال (١٣-٩)

*تشغيل قائمة متصلة* . نوسع فى هذا المثال بعض الأفكار التى سبق تقديمها فى مثال١٢- ٨ . وبصفة خاصة نقدم الآن برنامجا متداخلا بلغة البسكال ، يسمح لنا بإنتاج قائمة متصلة ، ويضيف عنصراً جديداً ويحذف أى عنصر . والبرنامج يستخدم القوائم لتسهيل استخدامه بواسطة مستفيدين غير ملمين بالبرمجة . ويوجد به جزء لعرض القائمة بترتيب عكسى ( أى يظهر آخر عنصر أولا ) بعد القيام باختيار أى عنصر من القائمة .

ويبدأ البرنامج بتعريفات النوع ، وتوضيحات المتغيرات التالية :

```
TYPE line = PACKED ARRAY [1..40] OF char;
     link = ↑personal;
     personal = RECORD
                    name : line;
                    next : link
                END;
VAR item,pointer,last : link;
    choice : 1..4;
    count : 0..40;
    itemname : line;
```

وتشبه هذه التعريفات والتوضيحات مثيلاتها المقدمة فى مثال١٣- ٧ ، بالرغم من أننا توسعنا فيها بعض الشئ لتشمل السمات الإضافية فى البرنامج الحالى .

اعتبر المجموعة الرئيسية الآن . إذا وضعنا إنتاج القائمة والعبارات الإجرائية المصاحبة لكل اختيار من اختيارات القائمة داخل إجراءات منفصلة . وأن المجموعة الإجرائية الرئيسية لاتحتوى إلا على مجموعة إشارات لهذه الإجراءات فى معظم تكوينها . وعلى هذا ... يمكننا أن نكتب مايلى :

```
BEGIN   (* main action block *)
   REPEAT
      menu;
      CASE choice OF
         1 : create;
         2 : add;
         3 : delete;
         4 :
      END
   UNTIL choice = 4
END.
```

لاحظ أننا اتصلنا أولا بالإجراء menu الذى يتسبب فى إدخال قيمة choice من لوحة المفاتيح ( يمكن أن يأخذ choice القيم الصحيحة ١ أو ٢ أو ٣ أو ٤ ) . نتصل بعد ذلك بإجراء إضافى يعتمد اختياره على القيمة التى تم تحديدها choice . وتتكرر هذه العملية حتى تتحدد قيمة ٤ لـ choice ، والتى تحدد توقف الشرط .

والآن دعنا نفحص الإجراء menu . إنه إجراء بسيط جدا ، يحتوى على عبارات مدخلات ومخرجات فقط .

```
PROCEDURE menu;

(* this procedure displays the main menu *)

BEGIN
   page;
   writeln('Main menu:');
   writeln;
   writeln('  1 - Create the linked list');
   writeln;
   writeln('  2 - Add a component');
   writeln;
   writeln('  3 - Delete a component');
   writeln;
   writeln('  4 - End');
   writeln;
   write('Enter your choice (1, 2, 3 or 4) -> ');
   readln(choice);
   writeln
END;
```

( يمكن وضع هذه العبارات ببساطة داخل المجموعة الإجرائية الرئيسية . وقد وضعت فى إجراء مستقل لتبسيط التنظيم الشامل لمنطق البرنامج ) .

والإجراءات المستخدمة فى إنتاج قائمة جديدة ، وإدخال اسم من لوحة المفاتيح ، وعرض القائمة الحالية تستخدم نفس المنطق بالضرورة مثل برنامج البسكال المعطى فى مثال١٣- ٨ . ( لاحظ على أية حال أن مجموعات العبارات المطلوبة لانتاج القائمة وعرضها نقلت من المجموعة الإجرائية الرئيسية إلى برامج فرعية subroutines فردية ) وعلى هذا ... يمكن إنتاج القائمة بالاتصال بالإجراء التالى :

```
PROCEDURE create;

(* this procedure creates the linked list

   item    -> the new (most recent) component
   pointer -> the preceding component

             ('item' points to 'pointer')      *)
BEGIN
   new(item);
   write('New name: ');
   readname(item↑.name);
   item↑.next := NIL;
   pointer := item;
   WHILE NOT ((item↑.name[1] IN ['E','e'])
         AND (item↑.name[2] IN ['N','n'])
         AND (item↑.name[3] IN ['D','d'])) DO
```

(تكملة البرنامج فى الصفحة التالية)

```
                    BEGIN
                       new(item);
                       write('New name: ');
                       readname(item↑.name);
                       item↑.next := pointer;
                       pointer := item
                    END;
          last := item↑.next;
          display
       END;
```

لاحظ أن آخر عنصر فى القائمة ملحق به المشير last . ويمكننا ذلك من إيجاد نهاية القائمة عندما نرغب فى عرضها ، وإضافة عنصر جديد ، أو حذف أحد عناصرها .

لاحظ أيضا أن القائمة الجديدة تعرض تلقائيا بعد إنتاجها .

الإجراء readname الذى يتم الاتصال به بواسطة الإجراء سالف الذكر ( وإجراءات أخرى أيضا ) . يكتب على النحو التالى :

```
PROCEDURE readname(VAR name : line);

(* this procedure reads a name into the computer *)

BEGIN
   FOR count := 1 TO 40 DO name[count] := ' ';
   count := 0;
   WHILE NOT eoln DO
      BEGIN
         count := count + 1;
         read(name[count])
      END;
   readln
END;
```

الاجراء فى هذا المثال يشبه تماما الإجراء الذى له نفس الاسم الموجود فى مثال١٣- ٨ . لاحظ أن الإجراء الحالى يستخدم على أية حال متغير سلسلة كمؤشر من نوع المتغير ، بينما الصيغة السابقة تستخدم مشيرا كمؤشر متغير . وتقدم المعالجة الحالية لمؤشر المتغير عمومية أكثر . وعلى هذا ... فهى تسمح بالاتصال بهذا الإجراء عن طريق عدة إجراءات أخرى ( وهى بالأسم delete , add , create ) .

والإجراء المستخدم لعرض القائمة يمكن أن يكتب على النحو التالى :

```
PROCEDURE display;
(* this procedure displays the complete list
   item    -> the current component to be displayed
   pointer -> the preceding component
             ('item' points to 'pointer')          *)
```

```
BEGIN
   writeln;
   pointer := last;
   WHILE pointer <> NIL DO
      BEGIN
         item := pointer;
         writeln(item↑.name);
         pointer := item↑.next
      END
END;
```

يبدأ هذا الإجراء بتحديد نهاية القائمة ( Pointer : = Last ) ثم يستمر بنفس الطريقة مثل البرنامج الموضح فى مثال١٣– ٨ .

ويسمح الإجراء التالى بإضافة عنصر إلى القائمة المتصلة التى تم إنتاجها .

```
   PROCEDURE add;
   (* this procedure allows one component
      to be added to the linked list
      item    -> the component to be added
      pointer -> component below (i.e., pointing to) the new component
                 ('pointer' points to 'item').                    *)
   BEGIN
      new(item);
      writeln;
      write('New name: ');
      readname(item↑.name);
      write('Place ahead of: (press RETURN if new item is last) ');
      readname(itemname);
      IF itemname[1] = ' '
         THEN BEGIN   (* insert at end *)
                 item↑.next := last;
                 last := item
              END
      ELSE BEGIN   (* find location, then insert *)
              pointer := last;
              WHILE (pointer↑.name <> itemname) AND
                    (pointer↑.next <> NIL) DO pointer := pointer↑.next;
              item↑.next := pointer↑.next;
              pointer↑.next := item
           END;
   display
END;
```

يتسبب هذا الإجراء أولا فى إدخال الاسم الجديد ( من لوحة المفاتيح ) ، وبعد ذلك تحديد موقعه . وبعد ذلك يتبع الإجراء المنطق الذى سبق ذكره فى هذا الفصل ، والذى يسمح بادخال عنصر جديد فى نهاية القائمة ( أى عند قمتها ) أو فى أى مكان آخر مناسب داخل القائمة . لاحظ أن القائمة الجديدة تعرض تلقائيا بعد إتمام الإضافة .

اعتبر الآن الإجراء المستخدم فى حذف عنصر من مكان محدد داخل القائمة . يمكن كتابة هذا الإجراء على النحو التالى : ( أنظر الصفحة القادمة ) .

```
PROCEDURE delete;

(* this procedure causes one component
   to be deleted from the linked list

   item    -> the component to be deleted
   pointer -> component above (i.e., pointing to)
                       the component to be deleted

              ('pointer' points to 'item')           *)

BEGIN
   writeln;
   write ('Name to be deleted: ');
   readname(itemname);
   IF last↑.name = itemname
      THEN BEGIN   (* delete last item *)
              item := last;
              last := item↑.next;
              dispose(item)
           END
      ELSE BEGIN   (* find item, then delete *)
              pointer := last;
              item := last↑.next;
              WHILE (item↑.name <> itemname) AND
                    (item↑.next <> NIL) DO
                 BEGIN
                    pointer := item;
                    item := item↑.next;
                 END;
              IF item↑.name = itemname THEN
                 BEGIN
                    pointer↑.next := item↑.next;
                    dispose(item)
                 END
           END;
   display
END;
```

لقد سبق ظهور التفاصيل المنطقية لهذا الإجراء فى هذا الفصل . لاحظ أن القائمة الجديدة تعرض تلقائيا بعد إتمام عملية الحذف .

وفيما يلى البرنامج كاملا .

```
PROGRAM linklist(input,output);

(* THIS IS A MENU-DRIVEN PROGRAM THAT USES POINTERS
   TO PROCESS A LINKED LIST OF NAMES *)

TYPE line = PACKED ARRAY [1..40] OF char;
     link = ↑personal;
     personal = RECORD
                   name : line;
                   next : link
                END;
```

(تكملة البرنامج فى الصفحة التالية)

```
VAR item,pointer,last : link;
    choice : 1..4;
    count : 0..40;
    itemname : line;

PROCEDURE readname(VAR name : line);

(* this procedure reads a name into the computer *)

BEGIN
   FOR count := 1 TO 40 DO name[count] := ' ';
   count := 0;
   WHILE NOT eoln DO
      BEGIN
         count := count + 1;
         read(name[count])
      END;
   readln
END;

PROCEDURE display;

(* this procedure displays the complete list

   item    -> the current component to be displayed
   pointer -> the preceding component

             ('item' points to 'pointer')          *)
BEGIN
   writeln;
   pointer := last;
   WHILE pointer <> NIL DO
      BEGIN
         item := pointer;
         writeln(item↑.name);
         pointer := item↑.next
      END
END;

PROCEDURE create;

(* this procedure creates the linked list

   item    -> the new (most recent) component
   pointer -> the preceding component

             ('item' points to 'pointer')      *)
BEGIN
   new(item);
   write('New name: ');
   readname(item↑.name);
   item↑.next := NIL;
   pointer := item;
   WHILE NOT ((item↑.name[1] IN ['E','e'])
         AND (item↑.name[2] IN ['N','n'])
         AND (item↑.name[3] IN ['D','d'])) DO
```

(تكملة البرنامج فى الصفحة التالية)

```
            BEGIN
               new(item);
              write('New name: ');
              readname(item↑.name);
              item↑.next := pointer;
              pointer := item
            END;
   last := item↑.next;
   display
END;

PROCEDURE add;

(* this procedure allows one component
   to be added to the linked list

   item    -> the component to be added
   pointer -> component below (i.e., pointing to) the new component

              ('pointer' points to 'item')                   *)

BEGIN
   new(item);
   writeln;
   write('New name: ');
   readname(item↑.name);
   write('Place ahead of: (press RETURN if new item is last) ');
   readname(itemname);
   IF itemname[1] = ' '
      THEN BEGIN   (* insert at end *)
              item↑.next := last;
              last := item
           END
      ELSE BEGIN   (* find location, then insert *)
              pointer := last;
              WHILE (pointer↑.name <> itemname) AND
                    (pointer↑.next <> NIL) DO pointer := pointer↑.next;
              item↑.next := pointer↑.next;
              pointer↑.next := item
           END;
   display
END;

PROCEDURE delete;

(* this procedure causes one component
   to be deleted from the linked list

   item    -> the component to be deleted
   pointer -> component above (i.e., pointing to)
                          the component to be deleted

              ('pointer' points to 'item')           *)
BEGIN
   writeln;
   write ('Name to be deleted: ');
   readname(itemname);
   IF last↑.name = itemname
      THEN BEGIN   (* delete last item *)
              item := last;
              last := item↑.next;
              dispose(item)
           END
```

(تكملة البرنامج فى الصفحة التالية)

```
      ELSE BEGIN   (* find item, then delete *)
              pointer := last;
              item := last↑.next;
              WHILE (item↑.name <> itemname) AND
                    (item↑.next <> NIL) DO
                 BEGIN
                    pointer := item;
                    item := item↑.next;
                 END;
              IF item↑.name = itemname THEN
                 BEGIN
                    pointer↑.next := item↑.next;
                    dispose(item)
                 END
           END;
   display
END;

PROCEDURE menu;

(* this procedure displays the main menu *)

BEGIN
   page;
   writeln('Main menu:');
   writeln;
   writeln('  1 - Create the linked list');
   writeln;
   writeln('  2 - Add a component');
   writeln;
   writeln('  3 - Delete a component');
   writeln;
   writeln('  4 - End');
   writeln;
   write('Enter your choice (1, 2, 3 or 4) -> ');
   readln(choice);
   writeln
END;

BEGIN   (* main action block *)
   REPEAT
      menu;
      CASE choice OF
         1 : create;
         2 : add;
         3 : delete;
         4 :
      END
   UNTIL choice = 4
END.
```

دعنا نستخدم هذا البرنامج فى إنتاج قائمة متصلة تحتوى على المدن التالية : Denver , Chicago , Boston , SanFransisco , Pittssburgh , New Yourk . وبعد ذلك نضيف عدة مدن إضافية ، ونحذف عدة مدن أخرى ، موضحين كل سمات البرنامج . وسوف نحفظ قائمة المدن فى ترتيب أبجدى فى هذا التمرين . وعى هذا ... فإننا ندخل القائمة الأولية من الخلف ، بحيث تطبع المدن بالترتيب الصحيح ( لاحظ أن SanFransisco ستكون فى واقع الأمر فى بداية القائمة ، وأن Boston ستكون فى النهاية ) .

وفيما يلى الجزء المتداخل ، وكالعادة موضوع خط تحت استجابات المستفيد .

```
Main menu:

   1 - Create the linked list

   2 - Add a component

   3 - Delete a component

   4 - End

Enter your choice (1, 2, 3 or 4) -> 1

New name: San Francisco
New name: Pittsburgh
New name: New York
New name: Denver
New name: Chicago
New name: Boston
New name: end

Boston
Chicago
Denver
New York
Pittsburgh
San Francisco


Main menu:

   1 - Create the linked list

   2 - Add a component

   3 - Delete a component

   4 - End

Enter your choice (1, 2, 3 or 4) -> 2
```

(تكملة البرنامج فى الصفحة التالية)

```
New name: Atlanta
Enter ahead of (press RETURN if last item):

Atlanta
Boston
Chicago
Denver
New York
Pittsburgh
San Francisco

Main menu:

  1 - Create the linked list

  2 - Add a component

  3 - Delete a component

  4 - End

Enter your choice (1, 2, 3 or 4) -> 2

New name: Seattle
Enter ahead of (press RETURN if last item): San Francisco

Atlanta
Boston
Chicago
Denver
New York
Pittsburgh
San Francisco
Seattle

Main menu:

  1 - Create the linked list

  2 - Add a component

  3 - Delete a component

  4 - End

Enter your choice (1, 2, 3 or 4) -> 3

Name to be deleted: New York

Atlanta
Boston
Chicago
Denver
Pittsburgh
San Francisco
Seattle
```

(تكملة البرنامج فى الصفحة التالية)

```
Main menu:

   1 - Create the linked list

   2 - Add a component

   3 - Delete a component

   4 - End

Enter your choice (1, 2, 3 or 4) -> 2

New name: Washington
Enter ahead of (press RETURN if last item): Seattle

Atlanta
Boston
Chicago
Denver
Pittsburgh
San Francisco
Seattle
Washington

Main menu:

   1 - Create the linked list

   2 - Add a component

   3 - Delete a component

   4 - End

Enter your choice (1, 2, 3 or 4) -> 3

Name to be deleted: Atlanta

Boston
Chicago
Denver
Pittsburgh
San Francisco
Seattle
Washington

Main menu:

   1 - Create the linked list

   2 - Add a component

   3 - Delete a component

   4 - End

Enter your choice (1, 2, 3 or 4) -> 2
```

(تكملة البرنامج فى الصفحة التالية)

```
New name: Dallas
Enter ahead of (press RETURN if last item): Chicago

Boston
Chicago
Dallas
Denver
Pittsburgh
San Francisco
Seattle
Washington

Main menu:

  1 - Create the linked list

  2 - Add a component

  3 - Delete a component

  4 - End

Enter your choice (1, 2, 3 or 4) -> 3

Name to be deleted: Washington

Boston
Chicago
Dallas
Denver
Pittsburgh
San Francisco
Seattle


Main menu:

  1 - Create the linked list

  2 - Add a component

  3 - Delete a component

  4 - End

Enter your choice (1, 2, 3 or 4) -> 4
```

## أسئلة مراجعة : Review Questions

(١) ماذا تعنى القائمة المتصلة ؟

(٢) ماهو المشير ؟ وماهى العلاقة بين المشيرات والقوائم المتصلة ؟

(٣) كيف تختلف القائمة المتصلة عن القائمة التتابعية – على سبيل المثال – الموجودة فى منظومة ذات بعد واحد ؟

(٤) كيف تعرف بداية القائمة المتصلة ؟

(٥) ماهى القائمة المتصلة الخطية ؟ (ب) وماهى القائمة المتصلة الدائرية ؟ وكيف تختلف القائمة المتصلة الدائرية عن القائمة المتصلة الخطية ؟

(٦) ماهى الشجرة ؟ كيف تختلف الأشجار عن القوائم المتصلة الخطية ؟ ( ب) وماهى الشجرة الثنائية ؟ وكيف تختلف الشجرة الثنائية عن بقية الأنواع الأخرى من الأشجار ؟

(٧) ماهى النقاط والفروع ؟ وأى نوع من أنواع تكوينات البيانات الذى تصاحبه النقاط والفروع ؟

(٨) ماهو نوع البيانات المستخدمة فى معالجة التكوينات المتصلة فى البسكال ؟

(٩) كيف يعرف النوع المشير فى البسكال ؟

(١٠) ماهو نوع تكوينات البيانات المستخدم بصفة عامة فى تعريف متغير إشارة ؟ وماسبب هذا الاختيار ؟

(١١) صف الطريقة التى ترتبط بها أنواع المشيرات مع أنواع متغيرات إشارة كل منهما مع الآخر .

(١٢) كيف توضح المتغيرات من النوع المشير ؟ وفى أى جزء من أجزاء برنامج البسكال تظهر هذه التوضيحات ؟

(١٣) حدد فى كلمات محددة ، كيف توضح متغيرات الإشارة ، وكيف يختلف مفهوم توضيحات متغيرات الإشارة عن توضيحات الأنواع الأخرى للمتغيرات ؟

(١٤) فى أى جزء من أجزاء برنامج البسكال يظهر توضيح متغيرات الإشارة ؟ قارن ذلك مع توضيحات أنواع المتغيرات الأخرى ، وحدد أسباب هذه الاختلافات .

(١٥) ماهو الغرض من عبارة new ؟ وماهو نوع المؤشر المطلوب ؟

(١٦) وضح العلاقة الموجودة بين متغير إشارة ، والمتغير المشير المناظر فيه .

(١٧) كيف يمكن تمييز اسم متغير الإشارة عن اسم المتغير المشير المناظر له ؟

(١٨) لخص القواعد المستخدمة فى تحديد متغير مشير لآخر .

(١٩) كيف يمكن جعل متغير مشير يشير إلى لاشئ؟

(٢٠) صف طريقتين مختلفتين لتحديد قيم لمتغيرات مشيرة . ماهو الفرض من كل نوع من نوعى التحديد هذين ؟

(٢١) لخص القواعد المستخدمة فى تحديد متغير إشارة لآخر .

(٢٢) هل يمكن تحديد حقول فردية موجودة داخل متغيرات إشارة لحقول أخرى ؟ ماهى القيود الموضوعة على هذا النوع من أنواع التحديد ؟

(٢٣) كيف تختلف مفاهيم تحديد أحد المتغيرات المشيرة لآخر عن تحديد أحد متغيرات الإشارة لآخر ؟

(٢٤) ماهى أنواع المقارنات التى يمكن إجراؤها بين المتغيرات المشيرة ؟ وماهى المؤثرات التى يمكن استخدامها فى هذا الفرض؟

(٢٥) ماهى أنواع المقارنات التى يمكن إجراؤها بين متغيرات الاشارة ؟ هل يمكن أداء مثل هذه المقارنات على محتويات متغيرات ؟ وهل يمكن أداؤها على حقول فردية ؟ ماهى القيود الموضوعة على مثل هذه المقارنات ؟

(٢٦) كيف تتصل العناصر الفردية داخل قائمة متصلة طبقا لقاعدة LIFO مع بعضها ؟

(٢٧) لخص المنطق المستخدم فى إنتاج عناصر جديدة داخل قائمة متصلة طبقا لقاعدة LIFO . فى أى ترتيب يتم إدخال العناصر ؟ وكيف تتصل العناصر مع بعضها ؟

(٢٨) لخص المنطق المستخدم فى عرض عناصر قائمة متصلة اتصال LIFO . فى أى ترتيب تظهر العناصر ؟

(٢٩) لخص المنطق المستخدم فى إدخال عناصر داخل بعض المواقع المحددة فى قائمة متصلة اتصال LIFO . كيف يعاد ضبط المشيرات ؟

(٣٠) ماهو الفرض من عبارة dispose ؟ وماهو نوع المؤشرات المطلوب ؟ قارن ذلك مع عبارة new .

(٣١) لخص المنطق المستخدم فى حذف عنصر من قائمة متصلة اتصال LIFO . كيف يعاد ضبط المشيرات ؟

## مسائل محلولة : Solved Problems

(٣٢) فيما يلى بعض تعريفات لأنواع مشيرات ، وأنواع متغيرات إشارة مناظرة لها .

(*a*)
```
TYPE next = ↑stockitem;
     stockitem = RECORD
                   stockno  : 1..9999;
                   quantity : integer;
                   nextitem : next
                 END;
```

```
(b) TYPE line = PACKED ARRAY [1..25] OF char;
         pointer = ↑customer;
         customer = RECORD
                        name   : line;
                        street : line;
                        city   : line;
                        next   : pointer
                     END;
(c) TYPE next = ↑part;
         part = RECORD
                   length    : real;
                   width     : real;
                   depth     : real;
                   nextpart  : next
                END;
(d) TYPE values = ARRAY [1..3] OF real;
         pointer = ↑part;
         part = RECORD
                   dimension : values;
                   next : pointer
                END;
```

(٣٣) فيما يلي عدة تعريفات نوع وتوضيحات متغيرات مصاحبة لها ، تشمل مشيرات ومتغيرات الإشارة المناظرة لها .

```
(a) TYPE next = ↑stockitem;
         stockitem = RECORD
                        stockno  : 1..9999;
                        quantity : integer;
                        nextitem : next
                     END;
    VAR firstitem,lastitem,thisitem : next;

    BEGIN
       .
       .
       new(thisitem);
       .
       .
    END.
(b) TYPE values = ARRAY [1..3] OF real;
         pointer = ↑part;
         part = RECORD
                   dimension : values;
                   next : pointer
                END;
    VAR workpart : pointer;

    BEGIN   (* main action part *)
       .
       .
       new(workpart);
       .
       .
    END.
```

(٣٤) فيما يلى تخطيطات هيكلية لعدة برامج بسكال ، كل برنامج يحتوى على استخدام مشيرات ومتغيرات إشارة .

(*a*)
```
PROGRAM sample(input,output);
TYPE line = PACKED ARRAY [1..25] OF char;
     pointer = ↑customer;
     customer = RECORD
                   name   : line;
                   street : line;
                   city   : line;
                   next   : pointer
                END;
VAR nameandaddress : pointer;

PROCEDURE readinput (VAR item : line);
(* read a 25-character string *)
BEGIN
   .
   .
   .
END;
BEGIN   (* main action block *)
   new(nameandaddress);
      .
      .
      .
   WITH nameandaddress↑ DO
      BEGIN
         readinput(name);
         readinput(street);
         readinput(city)
      END;
      .
      .
END.
```

لاحظ استخدام مكون WHILE - DO لتسهيل قراءة عناصر متغير الإشارة ( انظر فصل ١٠ ) . لاحظ أيضا أن المؤشرات الفعلية المستخدمة مع readinput ، هى متغيرات سلسلة كما هو مطلوب فى تعريف الإجراء .

(*b*)
```
PROGRAM sample(input,output);
TYPE line = PACKED ARRAY [1..25] OF char;
     pointer = ↑customer;
     customer = RECORD
                   name   : line;
                   street : line;
                   city   : line;
                   next   : pointer
                END;
VAR nameandaddress : pointer;

PROCEDURE readinput (VAR personal : pointer);
(* read the name, street and city *)
BEGIN
   .
   .
   .
END;
```

```
PROCEDURE writeoutput (personal : pointer);
(* display the name, street and city *)
BEGIN
   .
   .
   .
END;

BEGIN   (* main action block *)
   new(nameandaddress);
      .
      .
   readinput(nameandaddress);
      .
      .
   writeoutput(nameandaddress);
      .
      .
   dispose(nameandaddress);
   .
   .
END.
```

لاحظ أن المؤشرات الفعلية هي متغيرات من نوع المشير الآن : ( قارن ذلك مع المثال السابق ) لاحظ أيضا أن readinput يستخدم مؤشراً متغيراً ، بينما يستخدم writeoutput مؤشر قيمة (وكل منهما مؤشر من نوع المشير) .

(*c*)

```
PROGRAM sample(input,output);
TYPE values = ARRAY [1..3] OF real;
     pointer = ↑part;
     part = RECORD
              dimension : values;
              next : pointer
            END;
VAR firstpart,nextpart : pointer;
    area,volume : real;

BEGIN   (* main action statements *)
   new(nextpart);
      .
      .
   nextpart↑.dimension[1] := 1.0;
   nextpart↑.dimension[2] := 2.5;
   nextpart↑.dimension[3] := 0.3;
      .
      .
   nextpart↑.next := firstpart;
      .
      .
   WITH nextpart↑ DO
      BEGIN
         area := dimension[1] * dimension[2];
         volume := area * dimension[3];
         IF dimension[1] <> dimension[3] THEN . . .
                                         ELSE . . .
```

```
        END;
          .
          .
      IF nextpart↑.next = NIL THEN . . .
                              ELSE . . . ;
          .
          .
      dispose(nextpart);
          .
          .
   END.
```

العبارات الموجودة فى هذا المثال ليست مرتبطة مع بعضها ارتباطا منطقيا ، فهى موجهة ببساطة لتوضيح تحديدات ومقارنات مختلفة لمتغيرات مشيرة ، وعناصر متغيرات إشارة .

## مشاكل متكاملة : Supplementary Problems

(٣٥) اكتب أنواع تعريفات مناسبة ، أو توضيحات متغيرات لكل موقف من المواقف المذكورة أدناه :

أ – عرف نوع مشير اسمه link يسمى متغير الإشارة المناظر له customer ، ويحتوى على الخطوات التالية :

١ – name ( ٢٠ خانة )

٢ – acctno ( عدد صحيح يتراوح من ١ إلى ٩٩٩٩ )

٣ – balance ( كمية حقيقية ) .

٤ – next ( مشير يشير إلى السجل التالى ) .

ب – صف متغيرين من نوع المشير ، كل منهما من نفس نوع link . أعط اسمى lastcustomer , nextcustomer لهما .

ج – انتج ، ثم بعد ذلك احذف متغيرًا ديناميكيًا يناظر nextcustomer . لاحظ أن nextcustomer هو متغير من النوع المشير .

(٣٦) تشير المشاكل التالية إلى متغيرات من النوع المشير ، وإلى متغيرات الإشارة المناظرة لها ، والمعرفة فى المشكلة السابقة .

أ – حدد قيمة المشير فى nextcustomer↑ للمتغير المشير lastcustomer .

ب – هل تشير قيمة المؤشر فى nextcustomer↑ إلى لاشئ ؟ .

ج – حدد كل عنصر من عناصر nextcustomer↑ للعنصر الذى يناظره فى lastcustomer↑ .

د – حدد الموازنة 287.55 $ لـ nextcustomer الذى يكون رقم حسابه 1330 .

هـ – احصل على قيمة من قيم nextcustomer . name↑ ، وذلك عن طريق الاتصال بإجراء يسمى readinput .

( أدخل تخطيطا لـ readinput فى حلك ) .

و – خطط مكون WHILE - DO يستمر فى التنفيذ ، طالما أن قيمة nextcustomer ليست NIL .

(٣٧) فيما يلى عدة تخطيطات هيكلية لبرامج تحتوى على استخدام مشيرات ، ومتغيرات إشارة مناظرة لها . وبعض العبارات مكتوبة خطأ . عرف كل الأخطاء .

(*a*)
```
PROGRAM sample(input,output);
TYPE textline = PACKED ARRAY [1..80] OF char;
     link = ↑pointer;
     stockitem = RECORD
                   stockno : 1..9999;
                   stocktype : textline;
                   next : pointer
                 END;
VAR newitem,olditem : stockitem;
BEGIN
   .
   .
   .
END.
```

(*b*)
```
PROGRAM sample(input,output);
TYPE link = ↑player;
     player = RECORD
                ssn : PACKED ARRAY [1..9] OF char;
                position : PACKED ARRAY [1..12] OF char;
                age : 1..99;
                height : real;
                weight : real
              END;
VAR nextplayer,lastplayer : link;
BEGIN
   new(player);
      .
      .
   IF player↑.age < 40 THEN . . . ;
      .
      .
   IF nextplayer <> NIL THEN . . . ;
      .
      .
   END.
```

(*c*)
```
PROGRAM sample(input,output);
TYPE textline = PACKED ARRAY [1..30] OF char;
     link = ↑student;
     student = RECORD
                 name : textline;
                 next : link
               END;
VAR point,tag : link;
```

```
PROCEDURE readtext (VAR line : textline);
VAR count : 1..30;
BEGIN
   FOR count := 1 TO 30 DO line[count] := ' ';
   count := 1;
   REPEAT
      read(line[count]);
      count := succ(count)
   UNTIL eoln;
   readln
END;

BEGIN   (* main action block *)
   new(point);
   readtext(point↑.name);
   point↑.next := NIL;
   tag := point;
   REPEAT
      new(point);
      readtext(point↑.name);
      point↑.next := tag;
      tag := point
   UNTIL point↑.name[1] = '*';
   tag := point↑.next
END.
```

## Programming Problems مشاكل برمجة

(٣٨) عدل البرنامج المعطى فى مثال (١٣– ٨ ) ، بحيث إن كل عنصر داخل القائمة ( أى كل سجل ) يحتوى على اسم وعنوان الشارع والمدينة والولاية ورقم البريد . ضع الاسم على سطر خاص به ، وعنوان الشارع على سطر آخر ، وبقية المعلومات على سطر ثالث .

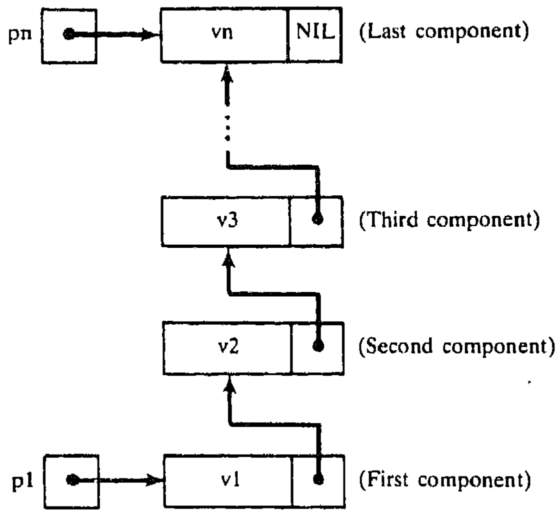
دع السجل يحتوى حقلين أول حقل ، عبارة عن منظومة ذات بعدين ، بها ٣ صفوف ، و ٨٠ عموداً ، ويمثل كل صف سطراً من أسطر النص . والحقل الثانى عبارة عن مشير إلى السجل التالى .

(٣٩) عدل البرنامج المعطى فى مثال (١٣– ٩ ) ، بحيث يمكن تطبيقه على كل من التكوينات المتصلة التالية :

أ – قائمة متصلة خطية ، بها فئتان من المشيرات ( فئة تشير فى الاتجاه الأمامى ، والفئة الأخرى تشير فى الاتجاه الخلفى )

ب – قائمة متصلة دائرية ( استخدم متغيراً مشيراً لتعريف بداية القائمة )

ج – قائمة متصلة خطية اتصال LIFO كما هو موضح فى شكل١٣– ٨




شكل١٣- ٨

د – شجرة ثنائية بعدد مستويات محدد ( لاحظ أن الطريقة المنطقية لحركة الشجرة مطلوبة ) .

(٤٠) اكتب برنامجا كاملا بلغة البسكال ، يسمح لك بإدخال شجرة العائلة الخاصة بك المعدة بالكمبيوتر . ابدأ بتحديد عدد الأجيال ( أى عدد المستويات ) ، ثم أدخل الاسماء والجنسيات لتعديل الشجرة ، ولإضافة أسماء جديدة ( انظر النقاط ) فى الشجرة . أدخل أيضا جزءًا لعرض من محتويات الشجرة تلقائيا بعد كل تجديد .

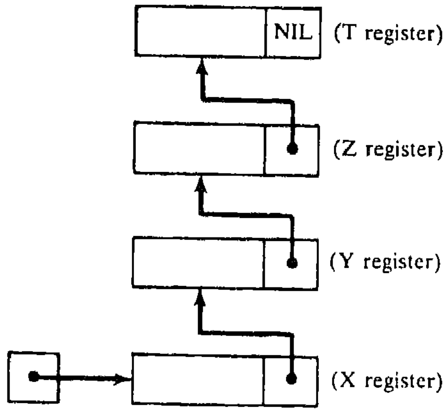
اختبر البرنامج ، مبتدئا بأول زوج من أجداد والديك ، وزوجًا من أجداد أجداد والديك .

(٤١) تستخدم إحدى الحاسبات RPN مخططا ، بحيث إن كل قيمة عددية جديدة تتبعها عملية تنفيذ على القيمة الجديدة والقيمة التى سبقتها . وحروف RPN هى اختصار Reverse Polish Notation التمييز البولندى للحركة . وعلى هذا ... فإضافة عددين ، وليكونا 3.3 , 4.8 ، يتطلب الضغط على المفاتيح التالية :

3.3 ⟨enter⟩
4.8 +

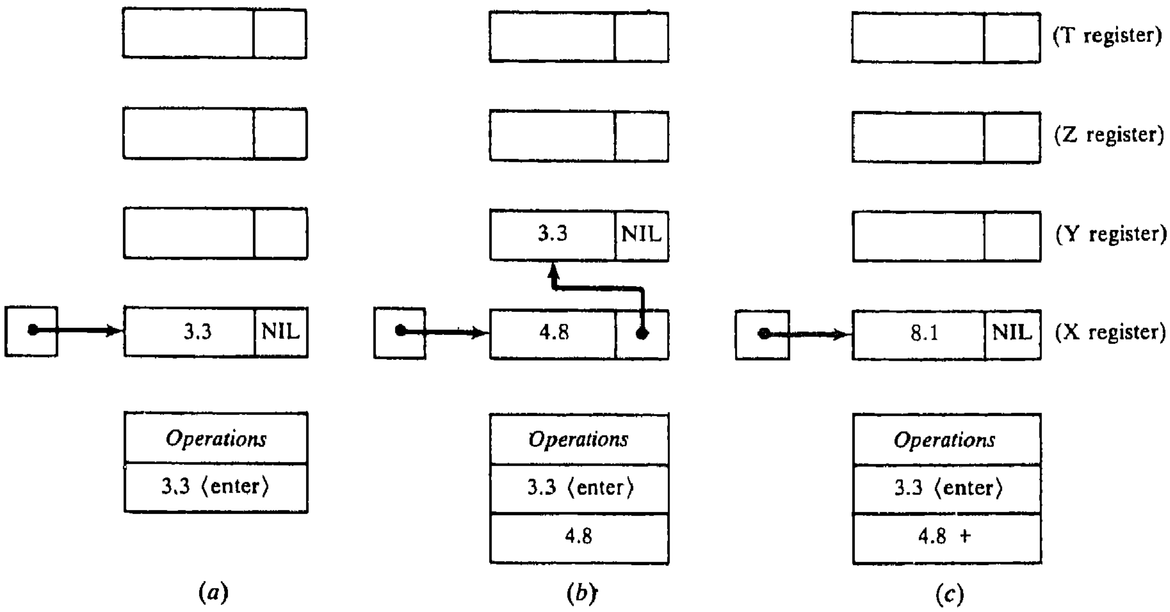
والمجموع 8.1 يعرض فى الحاسبة

تستخدم الحاسبات من هذا النوع حزمة تحتوى عادة على أربعة مسجلات ( أربعة عناصر ) كما هو موضح فى شكل١٣- ٩ . كل عدد جديد يتم إدخاله فى السجل X متسببا فى دفع كل القيم التى سبق إدخالها لأعلى فى الحزمة . فإذا ماكان المسجل T مشغولا من قبل ، فسوف يفقد العدد القديم ( حيث إن القيمة التى تدفع إليه من المسجل Z تحل محله ) .




شكل١٣- ٩

وتنفذ العمليات الحسابية دائما على الأعداد الموجودة فى المسجل X ، والمسجل Y . وتظهر نتيجة مثل هذه العمليات دائما فى المسجل X ، متسببه فى أن كل شئ فى المسجلات العلوية يسقط مستويا واحدا لأسفل ( أى تحـرك الحـزمة ) . هذه العملية موضحة فى الأشكال (c) 10 - 13 , (b) 10 - 13 (a) 10 - 13 بالنسبة لإجراء عملية جمع عددين سالفة الذكر .




شكل ١٣-١٠

اكتب برنامجا متداخلا بلغة البسكال ، يحاكى الحاسبة من نوع RPN . اعرض محتويات الحزمة بعد كل عملية ، كما فى الأشكال (c) 10 - 13 , (b) 10 - 13 (a) 10 - 13 ، على أن يكون البرنامج جزءًا لأداء كل من العمليات التالية :

| العملية | المفاتيح |
|---|---|
| إدخال بيانات جديدة | ( القيمة ) ( enter ) |
| الجمع | ( القيمة ) + |
| الطرح | ( القيمة ) − |
| الضرب | ( القيمة ) * |
| القسمة | ( القيمة ) / |

اختبر البرنامج مستخدما أى بيانات عددية من اختيارك .

# ملحق (أ)

## الكلمات المحجوزة

| | | | |
|---|---|---|---|
| AND | END | NIL | SET |
| ARRAY | FILE | NOT | THEN |
| BEGIN | FOR | OF | TO |
| CASE | FUNCTION | OR | TYPE |
| CONST | GOTO | PACKED | UNTIL |
| DIV | IF | PROCEDURE | VAR |
| DO | IN | PROGRAM | WHILE |
| DOWNTO | LABEL | RECORD | WITH |
| ELSE | MOD | REPEAT | |

# ملحق (ب)

## المعرفات القياسية

| | | | |
|---|---|---|---|
| abs | false | pack | sin |
| arctan | get | page | sqr |
| boolean | input | pred | sqrt |
| char | integer | put | succ |
| chr | ln | read | text |
| cos | maxint | readln | true |
| dispose | new | real | trunc |
| eof | odd | reset | unpack |
| eoln | ord | rewrite | write |
| exp | output | round | writeln |

# ملحق (جـ)

## الإجراءات القياسية

| الإجراء | الغرض منه | الإشارة إليه |
|---|---|---|
| dispose | حذف متغير ديناميكي ، أي حذف عنصر من عناصر قائمة متصلة | انظر القسم ٥ – ١٣ |
| get | نقل عناصر بيانات من ملف مدخلات إلى احتياطي ملف | انظر القسم ٥ – ١١ |
| new | إنتاج متغير ديناميكي ، أي إضافة عنصر إلى قائمة متصلة | انظر القسم ٣ – ١٣ |
| pack | ينتج عنه ضغط عناصر البيانات في المخزن ، أي داخل ذاكرة الكمبيوتر | انظر القسم ٤ – ٩ |
| page | إنتاج مخرجات جديدة ، تبدأ في بداية صفحة جديدة | انظر القسم ٧ – ٤ |
| put | نقل عناصر بيانات من احتياطي الملف إلى ملف مخرجات | انظر القسم ٣ – ١١ |
| read | قراءة عناصر بيانات من ملف مدخلات | انظر القسم ٢ – ٤ والقسم ٦ – ١١ |
| readln | قراءة عناصر بيانات من ملف مدخلات ، ثم النقل للسطر التالي | انظر القسم ٣ – ٤ |
| reset | إعداد الملف للقراءة | انظر القسم ٥ – ١١ |
| rewrite | إعداد الملف للكتابة | انظر القسم ٣ – ١١ |
| unpack | فك ضغط البيانات لتشغيل حيز أكبر من ذاكرة الكمبيوتر | انظر القسم ٤ – ٩ |
| write | كتابة بيانات في ملف مخرجات | انظر القسم ٥ – ٤ والقسم ٤ – ١١ |
| writeln | كتابة عناصر بيانات في ملف مخرجات ، ثم النقل للسطر التالي . | انظر القسم ٦ – ٤ |

# ملحق (د)

## الدوال القياسية

| الدالة | الغرض منها | نوع المؤشر | نوع النتيجة |
|---|---|---|---|
| abs (x) | حساب القيمة المطلقة | صحيح أو حقيقى | مثل x |
| arctan (x) | حساب الظل العكسى | صحيح أو حقيقى | حقيقى |
| chr (x) | تحديد الرمز الذى يمثله | صحيح أو حقيقى | حرفى |
| cos (x) | حساب جيب تمام x ( x بالتقدير الدائرى ) | صحيح أو حقيقى | حقيقى |
| eof (x) | تحديد ما إذا انتهى الملف أم لا | ملف | بوليان |
| eoln (x) | تحديد ما إذا أنتهى السطر أم لا | ملف | بوليان |
| exp (x) | حساب $e^x$ حيث $e = 2.7182818$ ، وهى أساس النظام الطبيعى للوغاريتمات | صحيح أو حقيقى | حقيقى |
| In (x) | حساب اللوغاريتم الطبيعى لـ x ( $x > 0$ ) | صحيح أو حقيقى | حقيقى |
| odd (x) | تحديد ما إذا كانت x فردية أم زوجية ( مع إعادة قيمة صحيح ( true ) إذا كانت x فردية ، وقيمة خاطئ false إذا كانت x زوجية ) | صحيح | بوليان |
| ord (x) | تحديد العدد العشرى الصحيح المستخدم فى عمل شفرة الحرف x . | حرفى | صحيح |
| pred (x) | تحديد مايسبق x . | صحيح أو حرفى أو بوليان | مثل نوع x |
| round (x) | تقريب قيمة x لأقرب رقم صحيح | حقيقى | صحيح |
| sin (x) | حساب جيب الزاوية x ( x بالتقدير الدائرى ) | صحيح أو حقيقى | حقيقى |
| sqr (x) | حساب مربع x . | صحيح أو حقيقى | مثل نوع x |
| sqrt (x) | حساب الجذر التربيعى لـ x ( $x >= 0$ ) | صحيح أو حقيقى | حقيقى |
| succ (x) | تحديد مايلى x | صحيح أو حرفى أو بوليان | مثل نوع x |
| trunc (x) | حذف الكسر العشرى من x | حقيقى | صحيح |

# ملحق ( هـ )

## المؤثرات

## Operators

مؤثرات البسكال ملخصة أدناه ، وذلك طبقا لأولويات التنفيذ الطبيعية ( الأعلى فالأقل )

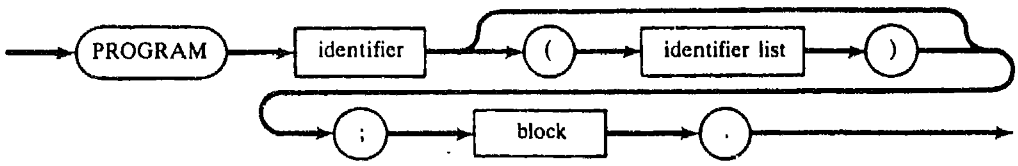
| أولوية التنفيذ | المؤثر ( المؤثرات ) |
|---|---|
| 1 ( الأعلى ) | NOT |
| 2 | * / DIV MOD AND |
| 3 | + - OR |
| 4 ( الأقل ) | = <> < <= > >= IN |

المؤثرات الموجودة فى نفس المستوى تنفذ كلها من اليمين إلى اليسار ، بغض النظر عن نوع المؤثر نفسه .

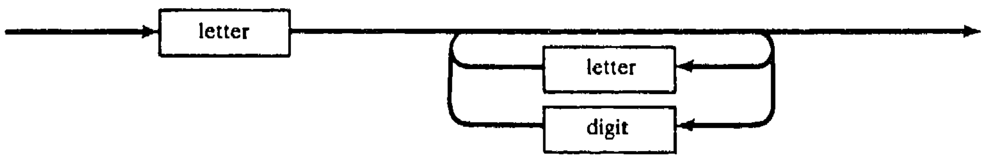
# ملحق ( و )

## رسومات التكوينات

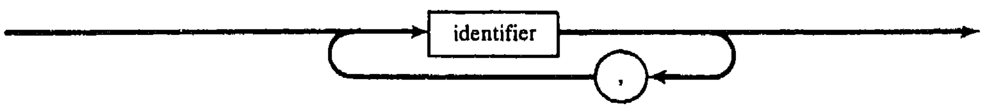
## Syntax Diagrams



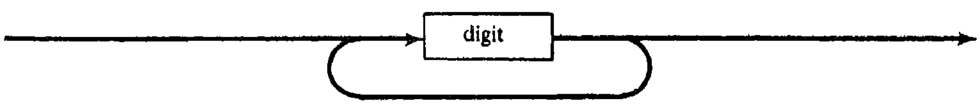

شكل ( و – ١ ) : برنامج



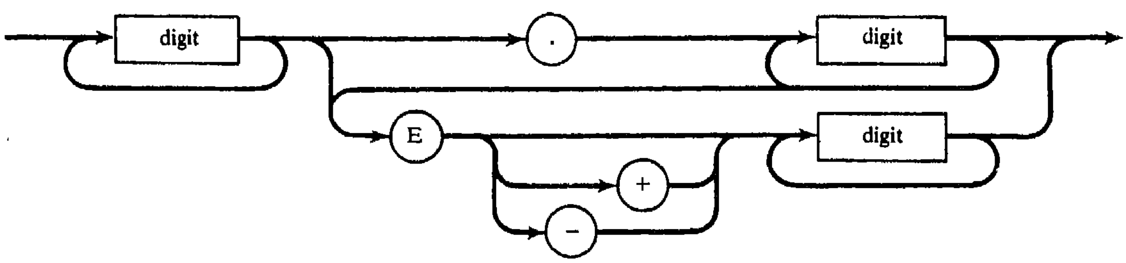

شكل ( و – ٢ ) : معرف



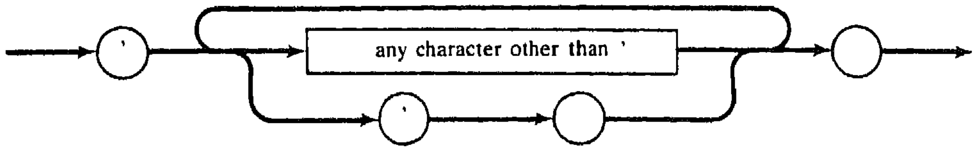

شكل ( و – ٣ ) : قائمة معرفات



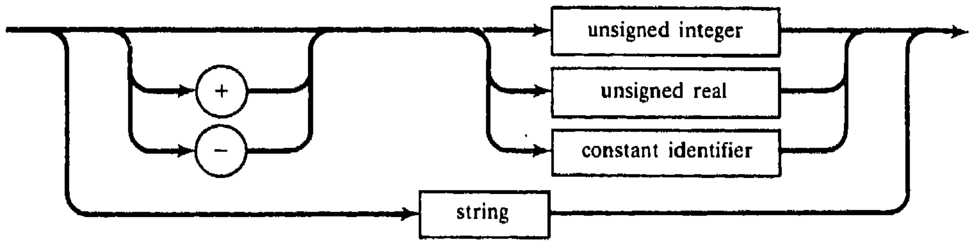

شكل ( و – ٤ ) : صحيح بدون إشارة



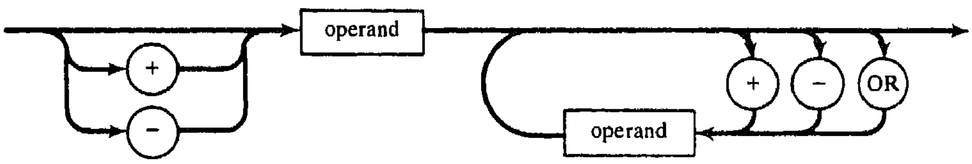

شكل ( و – ٥ ) : حقيقى بدون إشارة



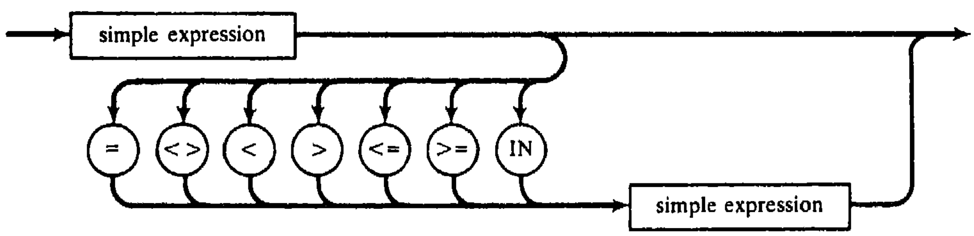

شكل (و – ٦) : سلسلة



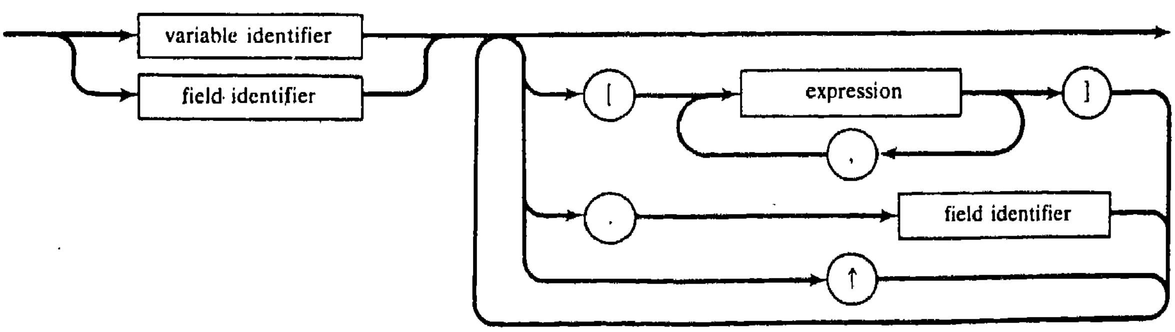

شكل (و – ٧) : ثابت



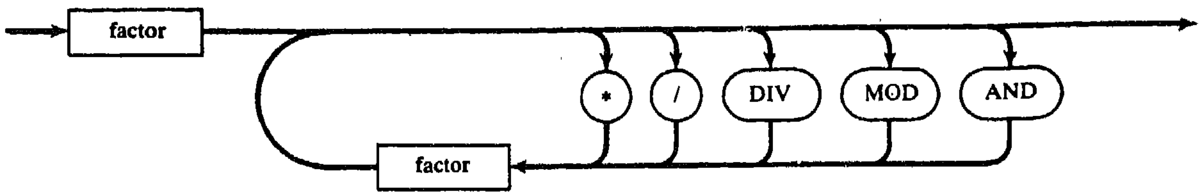

شكل (و – ٨) : تعبير بسيط



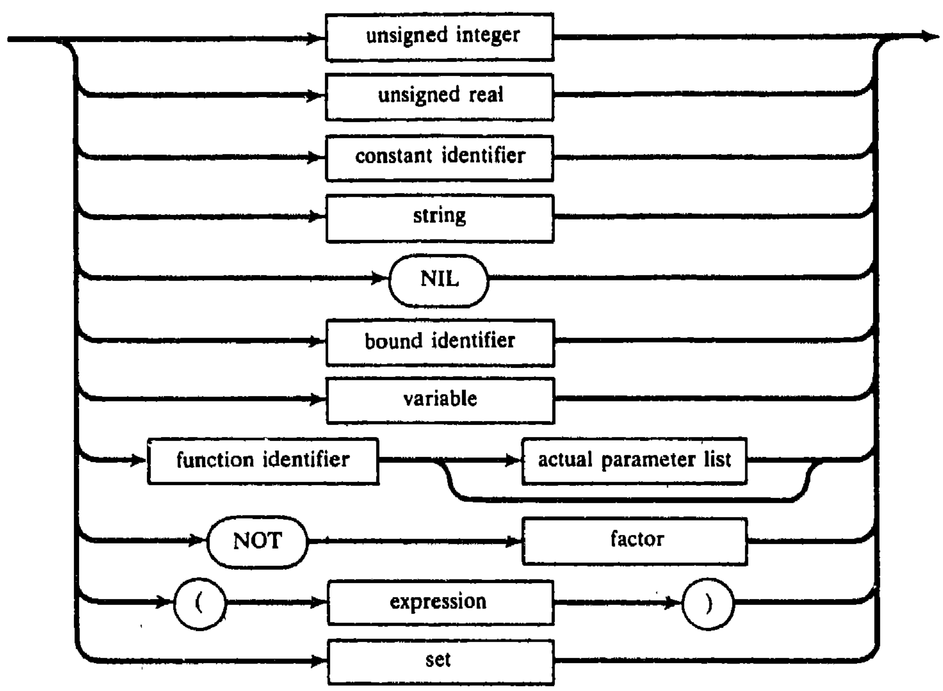

شكل (و – ٩) : تعبير



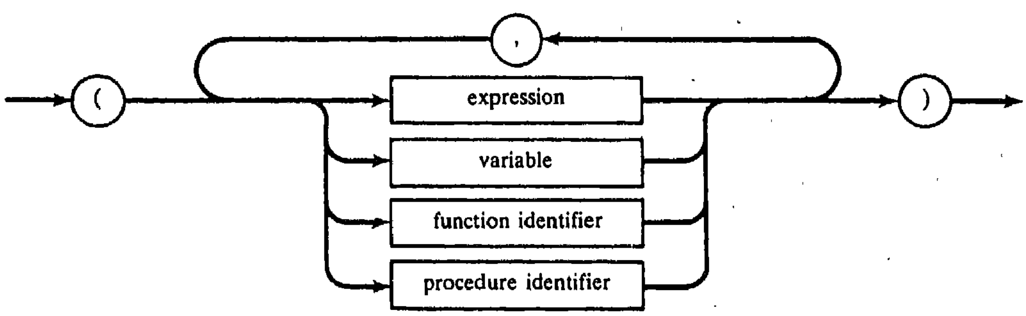

شكل (و – ١٠) : متغير



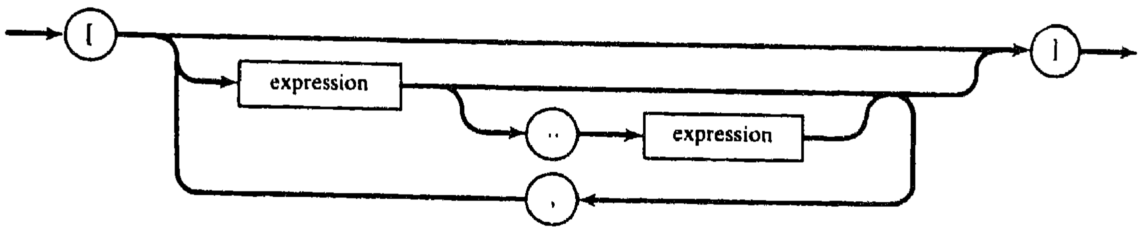

شكل ( و – ١١ ) : عنصر يجرى عليه عملية



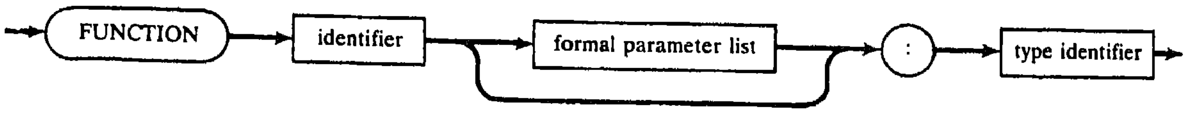

شكل ( و – ١٢ ) : معامل



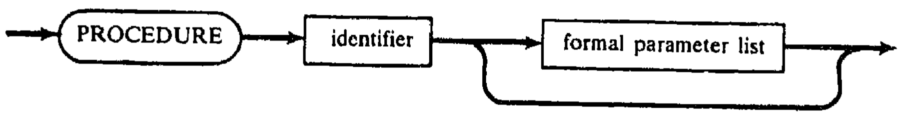

شكل ( و – ١٣ ) : قائمة مؤشرات فعلية



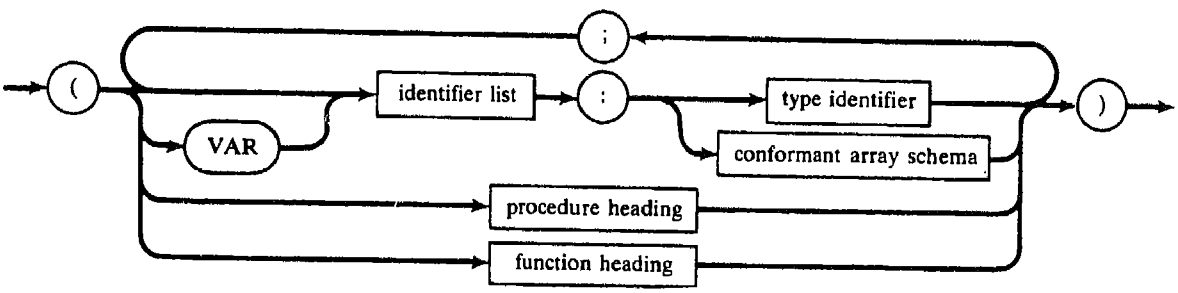

شكل ( و – ١٤ ) : فئة



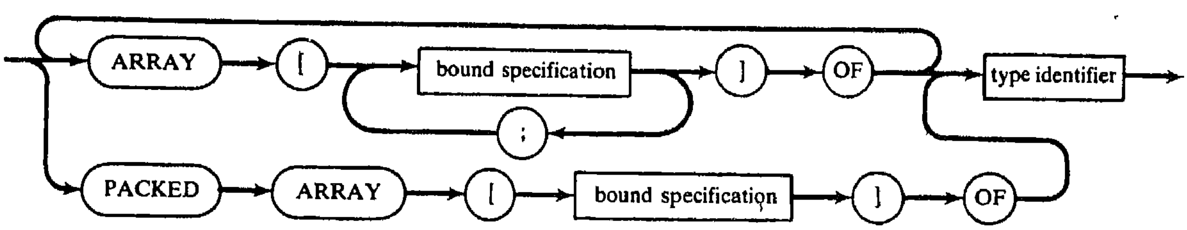

شكل ( و – ١٥ ) : عنوان دالة



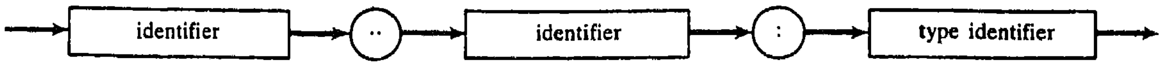

شكل ( و – ١٦ ) : عنوان إجراء



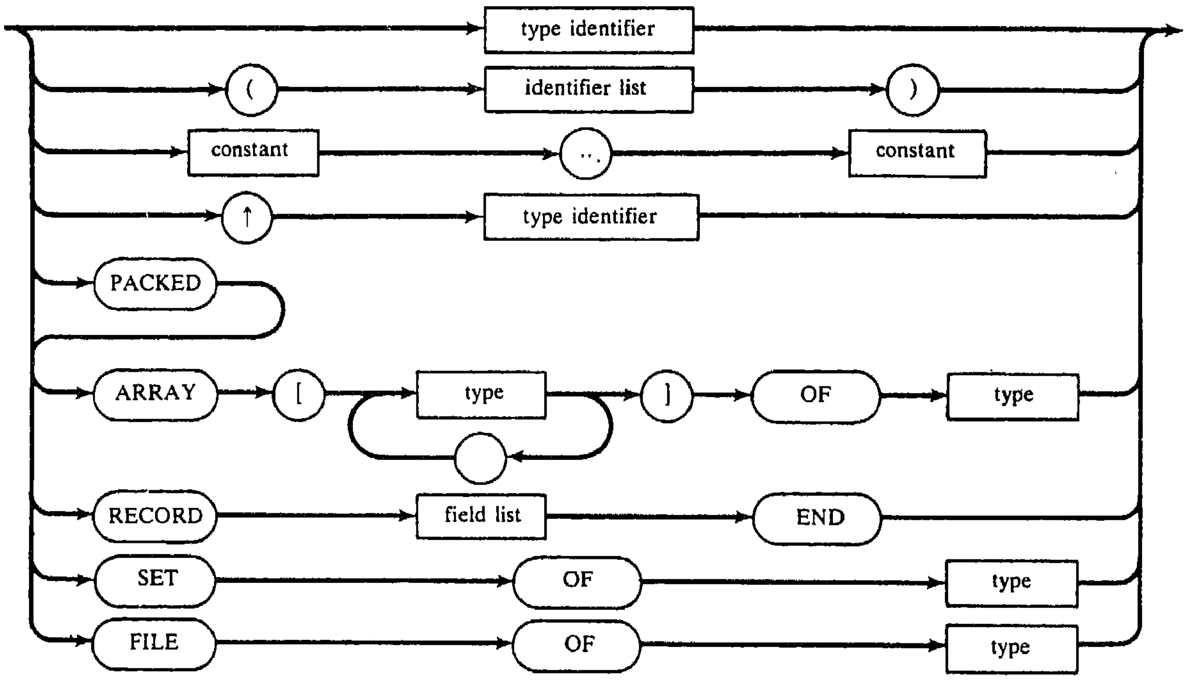

شكل ( و – ١٧ ) : قائمة مؤشرات رسمية



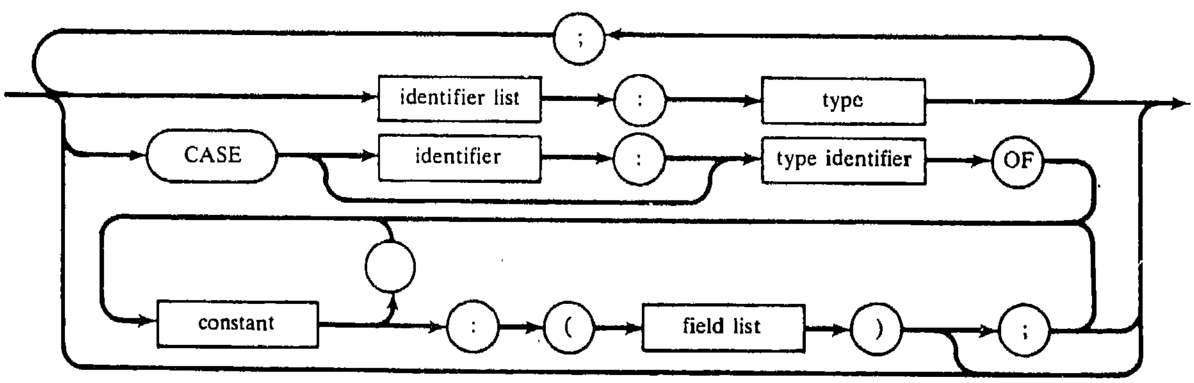

شكل ( و – ١٨ ) : مخطط منظومة



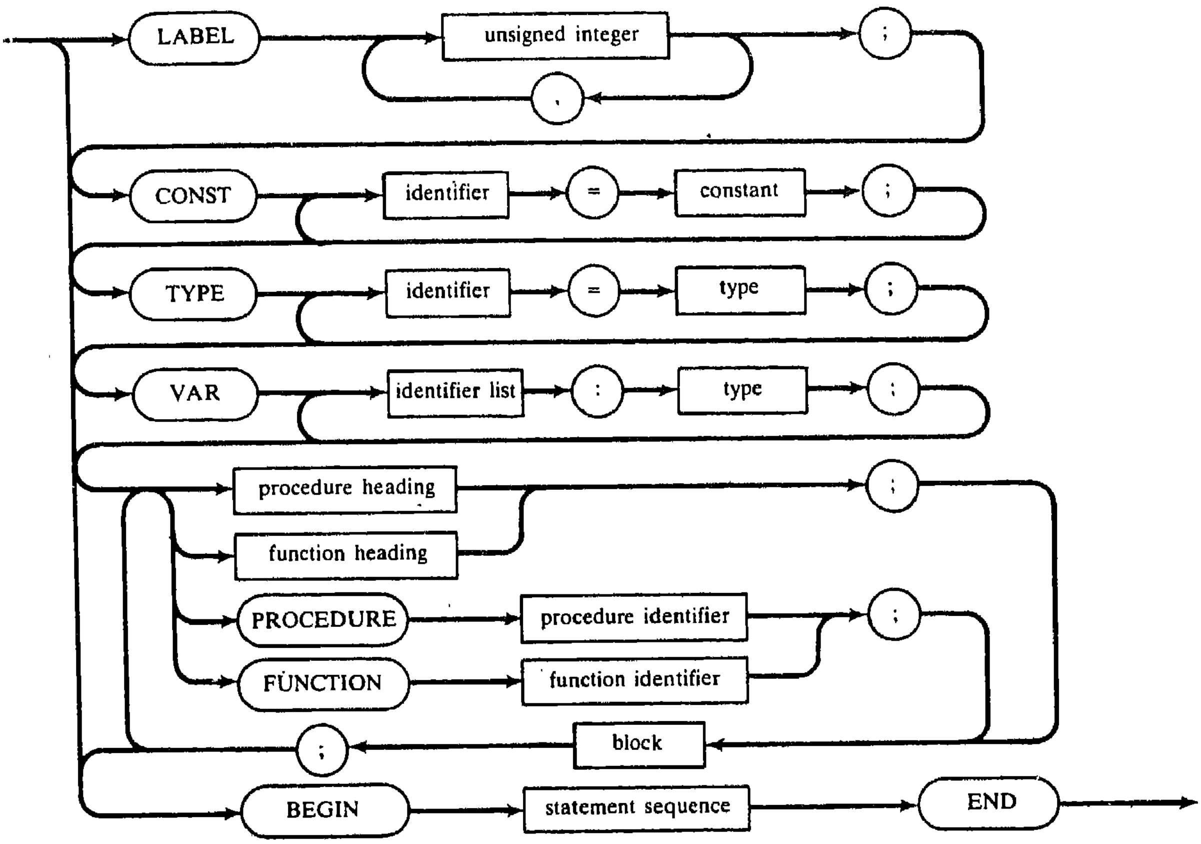

شكل ( و – ١٩ ) : مواصفة مربوطة



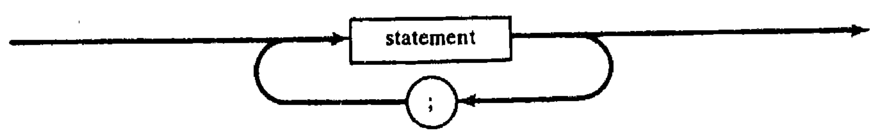

شكل ( و – ٢٠ ) : النوع



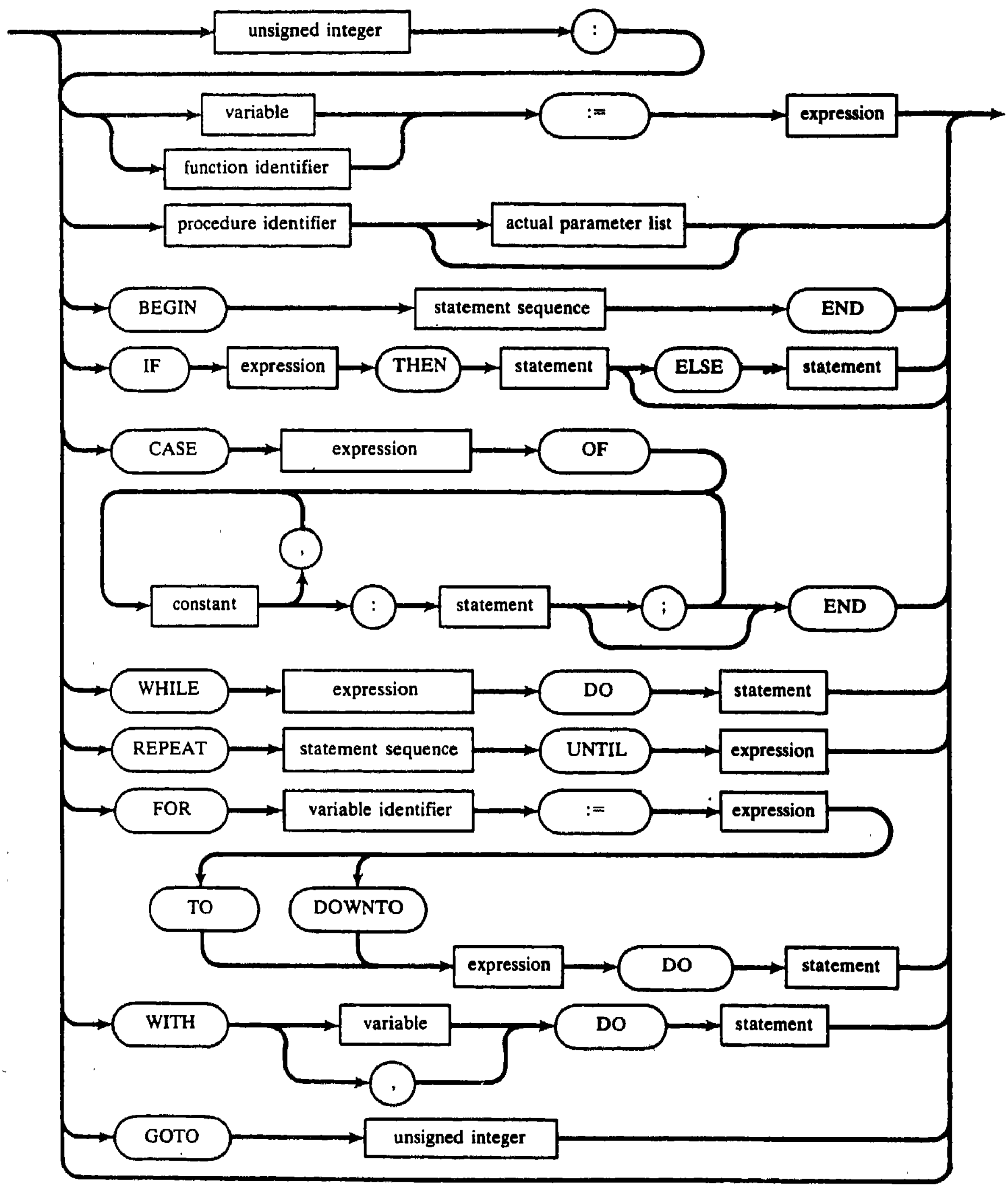

شكل ( و – ٢١ ) : قائمة حقول




شكل (و – ٢٢) : المجموعة




شكل (و – ٢٣) : تسلسل العبارة




شكل (و- ٢٤) : العبارة

# ملحق ( ز )

## فئة رموز ASCII

| ASCII Value | Character | ASCII Value | Character | ASCII Value | Character | ASCII Value | Character |
|---|---|---|---|---|---|---|---|
| 000 | NUL | 032 | blank | 064 | @ | 096 | ` |
| 001 | SOH | 033 | ! | 065 | A | 097 | a |
| 002 | STX | 034 | " | 066 | B | 098 | b |
| 003 | ETX | 035 | # | 067 | C | 099 | c |
| 004 | EOT | 036 | $ | 068 | D | 100 | d |
| 005 | ENQ | 037 | % | 069 | E | 101 | e |
| 006 | ACK | 038 | & | 070 | F | 102 | f |
| 007 | BEL | 039 | ' | 071 | G | 103 | g |
| 008 | BS | 040 | ( | 072 | H | 104 | h |
| 009 | HT | 041 | ) | 073 | I | 105 | i |
| 010 | LF | 042 | * | 074 | J | 106 | j |
| 011 | VT | 043 | + | 075 | K | 107 | k |
| 012 | FF | 044 | , | 076 | L | 108 | l |
| 013 | CR | 045 | - | 077 | M | 109 | m |
| 014 | SO | 046 | . | 078 | N | 110 | n |
| 015 | SI | 047 | / | 079 | O | 111 | o |
| 016 | DLE | 048 | 0 | 080 | P | 112 | p |
| 017 | DC1 | 049 | 1 | 081 | Q | 113 | q |
| 018 | DC2 | 050 | 2 | 082 | R | 114 | r |
| 019 | DC3 | 051 | 3 | 083 | S | 115 | s |
| 020 | DC4 | 052 | 4 | 084 | T | 116 | t |
| 021 | NAK | 053 | 5 | 085 | U | 117 | u |
| 022 | SYN | 054 | 6 | 086 | V | 118 | v |
| 023 | ETB | 055 | 7 | 087 | W | 119 | w |
| 024 | CAN | 056 | 8 | 088 | X | 120 | x |
| 025 | EM | 057 | 9 | 089 | Y | 121 | y |
| 026 | SUB | 058 | : | 090 | Z | 122 | z |
| 027 | ESC | 059 | ; | 091 | [ | 123 | { |
| 028 | FS | 060 | < | 092 | \ | 124 | \| |
| 029 | GS | 061 | = | 093 | ] | 125 | } |
| 030 | RS | 062 | > | 094 | ↑ | 126 | ~ |
| 031 | US | 063 | ? | 095 | — | 127 | DEL |

ملاحظة : أول ٣٢ رمز وآخر رمز ، هى رموز تحكم لايمكن طباعتها

# إجابات المشاكل المتكاملة

## الفصل الثانى

٥٠
- (a) - صحيح .
- (b) - صحيح .
- (c) - file هى كلمة محجوزة .
- (d) - صحيح .
- (e) - لايسمح بوجود فراغات .
- (f) - لايسمح باستخدام رموز غير الحروف والأرقام ، ( بعض صيغ البسكال تسمح برموز تحتها خط ) .
- (g) - يجب أن يكون أول رمز عبارة عن حرف أبجدى ، كما أن الشرطة غير مسموح بها أيضا .

٥١
- (a) - صحيح ( من النوع الحقيقى ) .
- (b) - لايسمح بوجود الفواصل .
- (c) - صحيح ( من النوع الحقيقى )
- (d) - قد كون قيمة الأس كبيرة جدا .
- (e) - يجب أن يظهر رقم صحيح قبل العلامة العشرية وبعدها .
- (f) - رمز غير مسموح به .
- (g) صحيح ( من النوع الحقيقى ) .
- (h) - صحيح ( من النوع الحقيقى ) قد يكون كبيرا بالنسبة لبعض أجهزة الكمبيوتر .
- (i) - صحيح ( من النوع الحقيقى ) .

٥٢
- (a) - صحيح .
- (b) - لايمكن استخدام علامتى التنصيص حول السلسلة .
- (c) - الفاصلة فى الناحية اليمنى غير موجودة .
- (d) - صحيح .
- (e) - صحيح .
- (f) صحيح .

٥٣

```
CONST month = 'july';
      fica = '123-45-6789';
      price = '$95.00';        (or price = 95.00;)
      gross = 2500.00;
      partno = 48837;
      bound = 0.00391;
```

٥٤

```
VAR period,status : char;
    terminal : boolean;
    index,row : integer;
    clearance : real;
```

٥٥ (*a*) - هذه عبارة تحديد ، وليست تعبير .
(*b*) – عددى أو بوليان ، وذلك طبقا لنوع البيانات المصاحبة للقيمة .
(*c*) – عددى .
(*d*) - بوليان .
(*e*) – غير مسموح بوجود مؤثرين متتاليين .
(*f*) - عددى .
(*g*) - بوليان .

٥٦ (*a*)- بسيط ( عبارة تحديد )
(*b*) – مرتب ( مركب ) .
(*c*) – مرتب ( تكرارى )
(*d*) - مرتب ( شرطى )
(*e*) – بسيط ( تحديد )
(*f*) - بسيط ( إجرائى )
(*g*) - بسيط ( نقل غير شرطى )
(*h*) – بسيط ( تحديد )

٥٧ يتم الاتصال بالإجراءات فى المشاكل 2.56(c) , 2.56(f) . ويتم الاتصال بالدالة فى المشكلة رقم 2.56(e) .

٥٨ (*a*) - يبدأ برنامج البسكال بكلمة البسكال المحجوزة ، يتبعها معرف ، ثم قائمة اختيارية بمعرفات موضوعة بين قوسين ، ثم تلى ذلك فاصلة منقوطة ، وتلى هذه العناصر مجموعة البرنامج ، أو صلب البرنامج .
(*b*) – يوضح هذا الرسم طرقا عديدة مختلفة لكتابة عدد حقيقى لإشارة واحدة له . وهى مايلى :
١ – رقم صحيح واحد أو أكثر ، يتبعه علامة عشرية ، وبعد ذلك رقم صحيح واحد أو أكثر .
٢ – رقم صحيح واحد أو أكثر ، يتبعه الحرف E ، وبعد ذلك رقم صحيح واحد أو أكثر ، وتظهر إشارة بين الحرف E والأرقام التى تليه ( هذه هى الصيغة الأسية ) .
٣ – خليط مما سبق ، أى رقم صحيح واحد أو أكثر ، تليه علامة عشرية ، ثم رقم صحيح ، تليه هذه العناصر ، والحرف يلى ذلك إشارة موجب أو إشارة سالب اختيارية ، ثم رقم صحيح واحد أو أكثر .
(*c*) - يبين هذا الشكل طرقا عديدة مختلفة لكتابة سلسلة ، وهى مايلى :
١ – رقم صحيح بدون إشارة تسبقه إشارة موجب أو إشارة سالب اختيارية .
٢ – رقم حقيقى بدون إشارة تسبقه إشارة موجب أو إشارة سالب اختيارية .
٣ – معرف ثابت تسبقه إشارة موجب أو إشارة سالب اختيارية .
٤ – سلسلة .

## الفصل الثالث

٤٠ (*a*) - حقيقى 0.3333333 - .
(*b*) – صحيح 2 - .
(*c*) – حقيقى 0.3333333 .
(*d*) - صحيح 14 - .
(*e*) – حقيقى 0.03 .
(*f*) - صحيح 3 .

٤١

(a) - خطأ .
(b) - صحيح .
(c) - صحيح .
(d) - صحيح .
(e) - صحيح .
(f) خطأ .
(g) - خطأ .
(h) - صحيح .

٤٢

| | | | | | |
|---|---|---|---|---|---|
| (a) | 4.667 | (i) | True | (p) | 2 |
| (b) | C | (j) | False | (q) | 2 |
| (c) | 101 | (k) | True | (r) | −2 |
| (d) | 9 | (l) | 2 | (s) | −2 |
| (e) | d | (m) | 3 | (t) | < |
| (f) | False | (n) | −2 | (u) | 64 |
| (g) | 11 | (o) | −3 | (v) | g |
| (h) | f | | | | |

٤٣

(b) الرمز المناظر للشفرة 67 فى نظام EBCDIC غير موجود فى جدول 3.2 .
(C) 133
(t) الرمز المناظر للشفرة 252 فى نظام EBCDIC غير موجود فى جدول 3.2 .
(u) 192

٤٤

(a) يتطلب DIV عنصراً صحيحاً .
(b) - صحيح .
(c) - لايمكن تنفيذ العمليات الحسابية على عناصر سلسلة .
(d) صحيح .
(e) - لايوجد مؤثر فى المقام .
(f) صحيح .
(g) يجب أن توضع عناصر بوليان بين قوسين .
(h) - صحيح .
(i) - المؤثرات غير متوفقة .
(j) - الأقواس غير متوازنة .
(k) - ينتج عن هذا التعبير قيمة تتعدى maxint .
(l) - غير مسموح بمؤثرات متتالية .
(m) - دالة Odd تتطلب مؤشراً صحيحاً .
(n) - صحيح .

٤٥

```
VAR gross,net,tax : real;
    employee : integer;
    status,sex : char;
    exempt : boolean;
```

## الفصل الرابع

٤٠ يتم إدخال بيانات المدخلات في سطرين ، على أن يوجد ثلاث كميات صحيحة ، وكميتان حقيقيتان فى أول سطر ، ويوجد أربعة رموز متتالية فى السطر الثانى .

وتحتوى المخرجات على ثلاثة أسطر ، بينها مسافات مزدوجة . يحتوى أول سـطر على قيم أولـى ثابـتيـن factor , flag . أما السطر الثانى ، فيحتوى على القيم الحالية للثلاثة متغيرات الصحيحة وللمتغيرين الحقيـقيين .

أما السطر الثالث ، فيحتوى على القيم الحالية للأربعة متغيرات من النوع الحرفى ومتغيرى بوليان .

وكل عناصر المخرجات الفردية تفصل عن بعضها ، فيما عدا الأربعة قيم المتتالية من النوع الحرفى الموجودة فى السطر الثانى .

٤١

```
(a) i1=1   i2=2   i3=3   r1=4.0   r2=5.0
    c1=b   c2=l   c3=u   c4=e
(b) i1=1   i2=2   i3=3   r1=4.0   r2=5.0
    c1=b   c2=(blank)   c3=(blank)   c4=(blank)
(c) i1=1   i2=2   i3=3   r1=4.0   r2=5.0
    c1, c2, c3 and c4 will be undefined
(d) i1=1   i2=2   i3=3   r1=4.0   r2=5.0
    c1=g   c2=r   c3=e   c4=e
```

٤٢

```
red 0.5000000E-02
(blank line)
     100    -200     -300 0.4004440E+03-0.5005550E+03
(blank line)
PINK  true false
```

٤٣

```
flag=red factor= 0.005
(blank line)
i1= 100 i2=-200 i3=-300 r1= 400.4 r2=-500.6
(blank line)
color=PINK b1= true b2=false
```

## تابع الفصل الرابع

٤٤

```
----------------------------------------
flag=red factor= 0.005      (top of page)
(blank line)
i1=  100 i2= -200 i3= -350
r1=  400.44 r2= -500.56
----------------------------------------
color=PINK                  (top of page)
(blank line)
b1=  true b2= false
```

٤٥

(*a*)
```
readln(i1,i2,i3,r1,r2,c1,c2,c3,c4);
```

(*b*)
```
readln(i1,r2,c3,c4);
```

(*c*)
```
readln(i1,i2,i3);
readln(r1,r2);
readln(c1,c2,c3,c4);
```

(*d*)
```
readln(i1);
readln(i2);
readln(i3);
readln(r1);
readln(r2);
readln(c1);
readln(c2);
readln(c3);
readln(c4);
```

(*e*)
```
write(flag,factor:6:3,i1:5,i2:5,i3:5,r1:9:3,r2:9:3);
writeln(' ',c1,' ',c2,' ',c3,' ',c4,b1,b2);
```
(Other solutions are also possible.)

(*f*)
```
write(flag,factor:8:2,i1:4,i2:4,i3:4,r1:8:2,r2:8:2);
writeln(' ',c1,' ',c2,' ',c3,' ',c4,b1:6,b2:6);
```

(*g*)
```
writeln(flag,factor:6);
writeln;
writeln(c1,' ',c2,' ',c3,' ',c4,b1:6,b2:6);
writeln;
writeln(i1:5,i2:5,i3:5,r1:9:3,r2:9:3);
```

(*h*)
```
writeln('flag=',flag,' factor=',factor);
writeln;
writeln('color=',c1,c2,c3,c4,' b1=',b1,' b2=',b2);
writeln;
write('i1=',i1:4,' i2=',i2:4,' i3=',i3:4);
writeln(' r1=',r1:8:3,' r2=',r2:8:3);
```

(*i*)
```
page;
writeln(flag,i1:5,r1:9:3,' ',c1,' ',c2,b1:6);
writeln;
writeln;
writeln(factor,i2:5,i3:5,r2:9:3,' ',c3,' ',c4,b2:6);
```
or
```
writeln('1',flag,i1:5,r1:9:3,' ',c1,' ',c2,b1:6);
writeln;
writeln('0',factor,i2:5,i3:5,r2:9:3,' ',c3,' ',c4,b2:6);
```

(*j*)
```
page;
writeln(flag,i1:5,r1:7:2,' ',c1,' ',c2,b1:7);
writeln;
writeln;
writeln(factor:7:2,i2:5,i3:5,r2:7:2,' ',c3,' ',c4,b2:7);
```

(*k*)
```
page;
writeln('flag=',flag);
writeln('factor=',factor:12);
writeln('i1=',i1:4);
writeln('i2=',i2:4);
writeln('i3=',i3:4);
writeln('r1=',r1:12);
writeln('r2=',r2:12);
writeln('c1=',c1);
writeln('c2=',c2);
writeln('c3=',c3);
writeln('c4=',c4);
writeln('b1=',b1:5);
writeln('b2=',b2:5);
```

٤٦

```
SUM= 60 PRODUCT=  6000
```

(Note that there is one blank space before 60, two before 6000.)

## – الفصل الخامس

(*a*)
```
PROGRAM hello(output);
(* print HELLO! at the beginning of a line *)
BEGIN
   writeln('HELLO!')
END.
```

(*b*)
```
PROGRAM friends(input,output);
(* interactive HELLO program *)
VAR c1,c2,c3,c4,c5,c6 : char;
BEGIN
   write('HI, WHAT''S YOUR NAME? ');
   readln(c1,c2,c3,c4,c5,c6);
   writeln;
   writeln;
   writeln('WELCOME ',c1,c2,c3,c4,c5,c6);
   writeln('LET''S BE FRIENDS!')
END.
```

(*c*) Noninteractive version:

```
PROGRAM celsius(input,output);
(* convert temperature from Fahrenheit to Celsius *)
VAR f,c : real;
BEGIN
   readln(f);
   c := (5/9)*(f-32);
   writeln('F=',f:7:2,'   C=',c:7:2)
END.
```

Interactive version:

```
PROGRAM celsius2(input,output);
(* convert temperature from Fahrenheit to Celsius *)
(* interactive version *)
VAR f,c : real;
BEGIN
   write('F= ');
   readln(f);
   c := (5/9)*(f-32);
   writeln('C=',c:7:2)
END.
```

(*d*) Noninteractive version:

```
PROGRAM piggybank(input,output);
(* piggy-bank problem *)
VAR n1,n2,n3,n4,n5 : integer;
    dollars : real;
BEGIN
   readln(n1,n2,n3,n4,n5);
   dollars := 0.5*n1 + 0.25*n2 + 0.1*n3 + 0.05*n4 + 0.01*n5;
   writeln;
   writeln('Half-dollars: ',n1:3);
   writeln('Quarters: ',n2:3);
   writeln('Dimes: ',n3:3);
   writeln('Nickels: ',n4:3);
   writeln('Pennies: ',n5:3);
   writeln;
   writeln('The piggy bank contains ',dollars:6:2,' dollars')
END.
```

Interactive version:

```
PROGRAM piggybank(input,output);
(* interactive piggy-bank problem *)
VAR n1,n2,n3,n4,n5 : integer;
    dollars : real;
BEGIN
   write('How many half-dollars? ');
   readln(n1);
   write('How many quarters? ');
   readln(n2);
   write('How many dimes? ');
   readln(n3);
   write('How many nickels? ');
   readln(n4);
   write('How many pennies? ');
   readln(n5);
   dollars := 0.5*n1 + 0.25*n2 + 0.1*n3 + 0.05*n4 + 0.01*n5;
   writeln;
   writeln('The piggy bank contains ',dollars:6:2,' dollars')
END.
```

(*e*)

```
PROGRAM sphere(input,output);
(* This program calculates the volume and area of a sphere, given the radius *)
CONST  pi = 3.14159;
VAR    radius,vol,area : real;
BEGIN
       writeln;
       writeln(' This program calculates the volume and area of a sphere.');
       writeln;
       write(' What is the radius?');
       readln(radius);

       (* Calculations.. *)

       area := 4.0*pi*sqr(radius);
       vol := area*radius/3.0;

       writeln;
       writeln(' The area is: ',area:8:2,' ,and the volume is: ',vol:7:2)
END.
```

(*f*)
```
PROGRAM mass(input,output);

(* This program calculates the mass of air in a tire *)

VAR     P,V,m,T : real;

BEGIN
        writeln;
        write(' This program calculates the mass of air in a tire');
        writeln(' using English units.');
        writeln;
        write(' What is the Pressure, in psi? ');
        readln(P);
        writeln;
        write(' What is the Volume, in Cubic Feet? ');
        readln(V);
        writeln;
        write(' What is the Temperature, in degrees Fahrenheit? ');
        readln(T);

        (* The calculations *)

        m := (P*V)/(0.37*(T+460.0));

        (* Write out the answer *)

        writeln;
        writeln(' The Mass of air, in Pounds, is: ',m:7:2)
END.
```

```
PROGRAM encode(input,output);

(* This program will encode a five-letter word. *)

VAR     a,b,c,d,e,a1,b1,c1,d1,e1 : char;

BEGIN
        writeln(' This program will encode a five-letter word.');
        writeln;
        write(' What word would you like encoded? ');
        readln(a,b,c,d,e);

        (* calculations *)

        a1 := chr(ord(a)-30);
        b1 := chr(ord(b)-30);
        c1 := chr(ord(c)-30);
        d1 := chr(ord(d)-30);
        e1 := chr(ord(e)-30);

        writeln;
        writeln(' The encoded word is: ',a1,b1,c1,d1,e1)
END.
```

(*h*)
```
PROGRAM decode(input,output);

(* This program decodes a five-letter word. *)

VAR     a,b,c,d,e,a1,b1,c1,d1,e1 : char;
```

```
BEGIN
      writeln(' This program will decode a five-letter word.');
      writeln;
      write(' What word would you like decoded? ');
      readln(a,b,c,d,e);

      (* calculations *)

      a1 := chr(ord(a)+30);
      b1 := chr(ord(b)+30);
      c1 := chr(ord(c)+30);
      d1 := chr(ord(d)+30);
      e1 := chr(ord(e)+30);

      writeln;
      writeln(' The decoded word is: ',a1,b1,c1,d1,e1)
END.
```

**الفصل السادس**

٤٢
(a) - صحيح .
(b) – صحيح .
(c) – خطأ ( أنواع بيانات مختلفة ) .
(d) - خطأ .
(e) – صحيح .
(f) - خطأ ( أنواع بيانات مختلفة )
(g) - صحيح .
(h) - خطأ .
(i) - خطأ .

٤٣
(a) لايمكن أن يظهر متغير من النوع الحقيقى فى تعبير بوليان .
(b) - صحيح .
(c) - لايمكن استخدام متغير من النوع الحقيقى كمتغير تحكم .
(d) صحيح .
(e) – صحيح على أن يكون b < c .
(f) القيم النسبية لكل من b , d غير متناسقة .
(g) صحيح .
(h) - اذا ماظل flag صحيحا true ، فتستمر دورة REPEAT - UNTIL فى التنفيذ لانهائيا ، وإلا فإن دورة WHILE - DO تنفذ مرة واحدة فقط .
(i) - صحيح .
(j) - القائم بالاختيار لايمكن أن يكون تعبيرا من النوع الحقيقى ، ولايمكن للعناوين أن تكون حقيقية .
(k) - لايمكن نقل التحكم إلى عبارة مركبة .
(l) - عناوين العبارات وعناوين الحالة لايمكن استخدامها بالتبادل .

٥٠

```
PROGRAM loan(input,output);

(* This program generates a monthly payment schedule for a loan *)

VAR principal,rate,interest,monthly_interest,payment,ipart,itot,ppart : real;
    month : integer;

BEGIN

   (* Read the input data *)

   writeln(' This program generates a monthly payment schedule for a loan.');
   writeln;
   writeln;
   write(' What is the amount of the loan? ');
   readln(principal);
   write(' What is the yearly interest rate, expressed as a percentage? ');
   readln(rate);
   interest := 0.01*rate;
   monthly_interest := interest/12.0;
   write(' What is the monthly payment? ');
   readln(payment);
```

```
   (* Carry out the calculations and write the results,
                                             on a month-by-month basis *)

   month := 1;
   writeln(' Payment   Interest   Principal   Unpaid Balance   Total Interest');
   WHILE principal*(1.0 + monthly_interest) >= payment DO
      BEGIN
         ipart := principal*monthly_interest;
         ppart := payment - ipart;
         principal := principal - ppart;
         itot := itot + ipart;
         write(' ',month:5,'     ',ipart:8:2,'   ',ppart:9:2,'     ',principal:9:2);
         writeln('         ',itot:9:2);
         month := month + 1
      END;
   ipart := principal*monthly_interest;
   ppart := ipart + principal;
   principal := 0.0;
   itot := itot + ipart;
   write(' ',month:5,'     ',ipart:8:2,'   ',ppart:9:2,'     ',principal:9:2);
   writeln('         ',itot:9:2)
END.
```

(*k*)
```
PROGRAM test(input,output);

(* This program averages sets of student examination scores *)

VAR avgtest,tottest,test : real;
    i,j,n,ntest,studid : integer;

BEGIN
   writeln(' This program averages n sets of student examination scores');
   writeln;

   REPEAT
      write(' How many students are there? ');
      readln(n);
      IF n < 0 THEN writeln(' Illegal value, please try again..')
   UNTIL n >= 0;

   REPEAT
      write(' How many exams for each student? ');
      readln(ntest);
      IF ntest < 0 THEN writeln(' Illegal value, please try again..')
   UNTIL ntest >= 0;
   writeln;

   FOR i := 1 TO n DO
```

```
        BEGIN
           writeln(' Student no. ',i:2);
           writeln;
           write(' Enter the student''s id: ');
           readln(studid);
           writeln;
           tottest := 0.0;
           writeln(' Enter the exam scores for this student');

           FOR j := 1 TO ntest DO
              BEGIN   (* enter and sum the exam scores *)
                 REPEAT
                    write(' Test No. ',j:2,': ');
                    readln(test);
                    IF (test < 0.0) OR (test > 100.0) THEN
                       writeln('Illegal score, please try again..');
                 UNTIL (test >= 0.0) AND (test <= 100.0);
                 tottest := tottest + test
              END;

           (* calculate and write out the average *)
           avgtest := tottest/ntest;
           writeln;
           writeln(' Average score: ',avgtest:6:2);
           writeln
        END
END.
```

(o)

```
PROGRAM pyramid(output);

(* This program creates a pyramid of numbers in an interesting pattern *)

VAR        a,b,c,d,e,f,g,val : integer;

BEGIN
   FOR a := -9 TO 0 DO
      BEGIN
         b := (-1)*a;          (* Change counter to positive values *)
         c := a + 10;
         write('                              ');

         FOR d := 1 TO b DO write(' ');
         val := c + c - 1;

         FOR e := c TO val DO
            BEGIN    (* Print left half of line, including center *)
               IF (e >= 10) THEN f := e - 10 ELSE f := e;
               write(f:1);
            END;
         val := c+c-2;
         FOR g := val DOWNTO c DO
            BEGIN    (* Print right half of line *)
               IF (g >= 10) THEN f := g - 10 ELSE f := g;
               write(f:1);
            END;
         writeln
      END
END.
```

(*p*)
```
PROGRAM plot(input,output);

(* This program generates a plot of a damped sinusoidal function *)

CONST star = '*';
      blank = ' ';
      dash = '-';
      line = '¦';
      linelength = 78;
VAR time,y,i,j,half : integer;

BEGIN
   writeln(' Generate a plot of the function exp(-0.1t) * sin(0.5t)');
   writeln;
   half := (linelength DIV 2);     (* Build a scale for the graph. *)

   FOR time := 0 TO 50 DO          (* Time 't' in seconds.  *)
      BEGIN
         y := ROUND((exp((-0.1)*(time)))*(sin(0.5*time))*(half DIV 1)) + half;
         IF time = 0 THEN        (* Plot the vertical axis *)
                      FOR i := 1 to linelength DO
                         IF i = y THEN write(star)
                                  ELSE write(dash)
                   ELSE        (* Plot the function *)
                      BEGIN
                         IF half > y THEN j := half
                                     ELSE j := y;
                          FOR i := 1 TO j DO
                             IF i = y THEN write(star)
                                      ELSE IF i = half THEN write(line)
                                                       ELSE write(blank)
                      END;
         writeln
      END
END.
```

## الفصل السابع

٣٥ (a)- لايمكن أن يستخدم اسم الإجراء كمتغير
(b)- صحيح .
(c)- صحيح .
(d)- يحتوى هذا المثال على عدة أخطاء :
١ - الدالة لايوجد لها نوع بيانات .
٢ - لايمكن أن يظهر اسم الدالة فى توضيح VAR داخلى .
٣ - المعرف الذى يمثل اسم الدالة لايمكن أن يحدد له قيمة جديدة .
(e) - المؤشرات الفعلية من نوع مختلف عن المؤشرات الرسمية .
(f)- صحيح .
(g)- المؤشرات الفعلية المناظرة لمؤشرات المتغير الرسمية لايمكن أن تكون تعبيرات ( لاحظ أن الإشارة إلى المتغير الشامل داخل الإجراء مسموح بها ) .
(h)- عدد المؤشرات الفعلية غير متساو مع عدد المؤشرات الرسمية المناظرة لها . كما أنه لايمكن جمع المتغيرات من النوع الحرفى مع بعضها .
(i) - صحيح ( لاحظ أن الدالة تستخدم متغيرات شاملة ) .
(j) - صحيح .
(k)- يتغير المتغير الشامل C كل مرة يتم الاتصال فيها بالدالة ( تأثير جانبى side effect ) .
(l)- لايمكن أن تمر الدوال القياسية كمؤشرات فى البسكال القياسى ( ISO ) .
(m)- صحيح .
(n)- يتم الاتصال بالدالة funct 1 ( داخل proc 1 ) قبل توضيحها .
(o)- يتم الاتصال بـ value ( داخل المجموعة الرئيسية ) خارج مدى توضيحها .
(p)- دالة الإعادة الذاتية لاتحتوى على شرط للإنهاء .

٣٦

(a) $y = x_n + \sum_{i=1}^{n-1} x_i$ or $y_n = x_n + y_{n-1}$

(b) $y = \frac{(-1)^n x^n}{n!} + \sum_{i=0}^{n-1} \frac{(-1)^i x^i}{i!}$ or $y_n = \frac{(-1)^n x^n}{n!} + y_{n-1}$

(c) $p = f_t * \prod_{j=1}^{t-1} f_j$ or $p_t = f_t * p_{t-1}$

٤٢

```
PROGRAM powerfunction(input,output);

(* This program calculates the power of a number.
   The function is y = x to the nth power, where n is an integer. *)

VAR answer : char;

FUNCTION y(x : real; n : integer) : real;

(* this function raises x to the nth power *)

VAR product : real;
    count : integer;
```

```
BEGIN
  (* form the product *)
    IF n = 0 THEN product := 1.0
            ELSE BEGIN
                   product := x;
                   IF abs(n) <> 1 THEN
                      FOR count := 2 TO abs(n) DO
                          product := product * x
                 END;

  (* test for negative exponent *)
    IF n >= 0 THEN y := product ELSE y := 1.0/product
END;

PROCEDURE exponent;

(* this procedure evaluates the power function and writes out the result *)

VAR power,x : real;
    n : integer;

BEGIN
   writeln;
   write(' What is the base value? ');
   readln(x);
   write(' What is the value of the exponent? ');
   readln(n);
   power := y(x,n);
   writeln;
   writeln(' ',x:6:2,' Raised to the ',n:3,' power is ',power:10:4)
END;

BEGIN   (* main action block *)
   REPEAT
      exponent;
      writeln;
      write(' Again (Y/N)? ');
      readln(answer)
   UNTIL ((answer = 'n') OR (answer = 'N'))
END.
```

٤٣

```
PROGRAM powerfunction(input,output);

(* This program calculates the power of a number.
   The function is y = x to the nth power.        *)

VAR answer : char;

FUNCTION y(x,n : real) : real;

(* this function raises x to the nth power *)

BEGIN
   y := exp(n*(ln(x)))
END;
```

```
PROCEDURE exponent;

(* this procedure evaluates the power function and writes out the result *)

VAR power,x,n : real;

BEGIN
   REPEAT
      writeln;
      write(' What is the base value? ');
      readln(x);
      IF (x < 0.0) THEN   (* write error message *)
         BEGIN
            writeln;
            writeln(' ERROR: The base value is negative.  Try again');
            writeln
         END
   UNTIL x >= 0.0;
   write(' What is the value of the exponent? ');
   readln(n);
   power := y(x,n);
   writeln;
   writeln(' ',x:6:2,' Raised to the ',n:6:2,' power is ',power:10:4)
END;
BEGIN   (* main action block *)
   REPEAT
      exponent;
      writeln;
      write(' Again (Y/N)? ');
      readln(answer)
   UNTIL ((answer = 'n') OR (answer = 'N'))
END.
```

(*g*) ٤٩

```
PROGRAM test(input,output);

(* This program averages sets of student examination scores *)

VAR avgtest,tottest : real;
    i,n,ntest : integer;


PROCEDURE readinput(n,ntest : integer; VAR tottest : real);

(* this procedure enters the input data for each student *)

VAR test : real;
    j,studid : integer;

BEGIN
   writeln(' Student no. ',i:2);
   writeln;
   write(' Enter the student''s id: ');
   readln(studid);
   writeln;
   tottest := 0.0;
   writeln(' Enter the exam scores for this student');
   (* enter and sum the exam scores for each student *)

   FOR j := 1 TO ntest DO
```

```
        BEGIN
           REPEAT
              write(' Test No. ',j:2,': ');
              readln(test);
              IF (test < 0.0) OR (test > 100.0)
                 THEN writeln('Illegal score, please try again..')
           UNTIL (test >= 0.0) AND (test <= 100.0);
           tottest := tottest + test
        END
END;


FUNCTION average(total : real; m : integer) : real;

(* this function determines an average from a total of m numbers *)

BEGIN
   average := total/m
END;

PROCEDURE writeanswer(avg : real);

(* this procedure writes out the average exam score for a particular student *)

BEGIN
   writeln;
   writeln(' Average score:   avg:6:2);
   writeln
END;

BEGIN   (* main action block *)
   writeln(' This program averages n sets of student examination scores');
   writeln;

   REPEAT
      write(' How many students are there? ');
      readln(n);
      IF n < 0 THEN writeln(' Illegal value, please try again..')
   UNTIL n >= 0;

   REPEAT
      write(' How many exams for each student? ');
      readln(ntest);
      IF ntest < 0 THEN writeln(' Illegal value, please try again..')
   UNTIL ntest >= 0;
   writeln;

   FOR i := 1 TO n DO
      BEGIN
         readinput(n,ntest,tottest);
         avgtest := average(tottest,ntest);
         writeanswer(avgtest)
      END
END.
```

٥٠

(c)
```
PROGRAM loan(input,output);

(* This program generates a monthly payment schedule for a loan *)

VAR principal,rate,interest,monthly_interest,amount : real;
   month,months : integer;

FUNCTION POWER(y : real; x : integer) : real;

(* this function raises y to the x power *)

BEGIN
  power := exp(x*(ln(y)))
END;

FUNCTION payment(principal,monthly_interest : real;
                                       months : integer) : real;

(* this function determines the monthly payment *)

VAR quantity : real;


BEGIN
  quantity := power((1 + monthly_interest),months);
  payment := principal * monthly_interest * quantity / (quantity - 1)
END;

PROCEDURE readinput(VAR principal,monthly_interest : real;
                                        VAR months : integer);
(* this procedure reads the input data *)
VAR rate,interest : real;
    years : integer;
BEGIN
  writeln;
  write(' What is the amount of the loan? ');
  readln(principal);
  write(' What is the yearly interest rate, expressed as a percentage? ');
  readln(rate);
  interest := 0.01*rate;
  monthly_interest := interest/12.0;
  write(' What is the duration of the loan, in years? ');
  readln(years);
  months := 12*years
END;

PROCEDURE writeoutput(principal,monthly_interest,amount : real;
                                                 months : integer);

(* this procedure generates and writes out a
                              month-by-month payment schedule *)

VAR ipart,itot,ppart : real;
    month : integer;
```

```
BEGIN
   write(' Payment   Interest   Principal   Unpaid Balance');
   writeln('   Total Interest');
   FOR month := 1 TO months DO
      BEGIN
         ipart := principal * monthly_interest;
         ppart := amount - ipart;
         principal := principal - ppart;
         itot := itot + ipart;
         write(' ',month:5,'     ',ipart:8:2,'   ',ppart:9:2);
         writeln('     ',principal:9:2,'          ',itot:9:2)
      END
END;

BEGIN   (* main action block *)
   writeln(' This program generates a monthly payment schedule for a loan.');
   writeln;

   (* read the input data *)

   readinput(principal,monthly_interest,months);

   (* calculate the monthly payment *)

   amount := payment(principal,monthly_interest,months);
   write(' Monthly payment: ',amount:9:2);
   writeln;

   (* write the month-by-month payment schedule *)

   writeoutput(principal,monthly_interest,amount,months)
END.
```

## الفصل الثامن

٢١ (*a*)- Miami .

(*b*) – Phoenix .

(*c*)– 6 .

(*d*)- 4 .

(*e*) – 4 .

(*f*) - صحيح .

(*g*) - خطأ .

٢٢ (*a*) صحيح .

(*b*) - التوضيحات موجودة فى ترتيب خاطئ . وأنواع البيانات غير متناسقة فى أول عبارة تحديد .

(*c*) - البيانات المتعددة لايمكن أن توجد فى عبارة read ، أو عبارة write .

(*d*) - صحيح .

(*e*) – صحيح .

٢٣

```
PROGRAM dates(input,output);

(* This program determines the number of days a person has been alive *)

TYPE day = 1..31;
     month = 1..12;
     year = 1960..2100;
     numberofdays = 0..maxint;

VAR dd : day;
    mm : month;
    yy : year;
    birthday,today,days : numberofdays;
    n,a,m,e,s : char;

FUNCTION daysbeyond1960(mm : month; dd : day; yy : year) : numberofdays;

(* This function counts the days from January 1, 1960 to a given date *)

VAR n : numberofdays;
BEGIN
   n := trunc(30.42*(mm - 1)) + dd;
   IF mm = 2 THEN n := n + 1;
   IF (mm > 2) AND (mm < 8) THEN n := n - 1;
   IF (yy MOD 4 = 0) AND (mm > 2) THEN n := n + 1;
   IF (yy-1960) DIV 4 > 0 THEN n := n + 1461*((yy-1960) DIV 4);
   IF (yy-1960) MOD 4 > 0 THEN n := n + 365*((yy-1960) MOD 4) + 1;
   daysbeyond1960 := n
END;

PROCEDURE header(VAR n,a,m,e,s : char);

(* This procedure generates the conversational input *)

BEGIN
   write(' Please enter your first name (5 letters): ');
   readln(n,a,m,e,s);
   writeln;
   writeln(' Hello, ',n:1,a:1,m:1,e:1,s:1);
   write(' This program will calculate the number of days ');
   writeln(' you have been alive');
   writeln
END;
BEGIN   (* main action block *)
   header(n,a,m,e,s);
   write(' When were you born, ',n:1,a:1,m:1,e:1,s:1,'? (mm dd yyyy) ');
   readln(mm,dd,yy);
   birthday := daysbeyond1960(mm,dd,yy);
   writeln;
   write(' What is today''s date? (mm dd yyyy) ');
   readln(mm,dd,yy);
   today := daysbeyond1960(mm,dd,yy);
   days := today - birthday + 1;
   writeln;
   writeln(' ',n:1,a:1,m:1,e:1,s:1,', you have been alive ',days:5,' days.')
END.
```

٢٤

```
PROGRAM dayoftheweek(input,output);

(* This program determines the day of the week for any given date *)

TYPE day = 1..31;
     month = 1..12;
     year = 1960..2100;
     numberofdays = 0..maxint;

VAR dd : day;
    mm : month;
    yy : year;
    n,a,m,e,s : char;

FUNCTION daysbeyond1960(mm : month; dd : day; yy : year) : numberofdays;

(* This function counts the days from January 1, 1960 to a given date *)

VAR n : numberofdays;
BEGIN
   n := trunc(30.42*(mm - 1)) + dd;
   IF mm = 2 THEN n := n + 1;
   IF (mm > 2) AND (mm < 8) THEN n := n - 1;
   IF (yy MOD 4 = 0) AND (mm > 2) THEN n := n + 1;
   IF (yy-1960) DIV 4 > 0 THEN n := n + 1461*((yy-1960) DIV 4);
   IF (yy-1960) MOD 4 > 0 THEN n := n + 365*((yy-1960) MOD 4) + 1;
   daysbeyond1960 := n
END;

PROCEDURE readinput(VAR mm : month; VAR dd : day;
                    VAR yy : year;  VAR n,a,m,e,s : char);

(* This procedure generates the conversational input *)

BEGIN
   write(' Please enter your first name (5 letters): ');
   readln(n,a,m,e,s);
   writeln;
   writeln(' Hello, ',n:1,a:1,m:1,e:1,s:1);
   writeln(' This program will calculate the day of the week');
   writeln(' for any given date beyond January 1, 1960');
   writeln;
   write(' ',n:1,a:1,m:1,e:1,s:1,', please enter the date (mm dd yyyy) : ');
   readln(mm,dd,yy)
END;
```

## الفصل التاسع

٤٤

(*a*) - صحيح .

(*b*) - صحيح .

(*c*) - صحيح .

(*d*) - الحدود الدنيا والعليا لأول فهرس موجودة فى ترتيب خاطئ .

(*e*) - الفهرس لايمكن أن يكون حقيقيا .

(*f*) - الفهرس لايقع فى المدى الصحيح .

(*g*) - صحيح .

(*h*) - مواصفات الفهرس فى عبارة التحديد غير متناسقة مع توضيح المنظومة .

(*i*) - عنصر البيانات المحدد للعنصر المنظومة من نوع خاطئ .

(*j*) - الفهارس فى عبارة التحديد من نوع بيانات خطأ .

(*k*) - صحيح .

(*l*) لايمكن أن تظهر محتويات المنظومة كعناصر داخل تعبير عددى .

(*m*) صحيح ( لأن name هو متغير سلسلة ) .

(*n*) - لايمكن أن تظهر محتويات المنظومة فى عبارات write , read ( وظهور متغير سلسلة فى عبارة write هو استثناء ) .

(*o*) - صحيح .

(*p*) - تكوين خاطئ لعبارات unpack , pack ( فالفهارس مكتوبة بطريقة خاطئة ، كما أن قيمها كبيرة جدا ) .

(*q*) - لايمكن مقارنة سلاسل لها أطوال مختلفة ولايمكن دمج متغيرات سلاسل ، أى لايمكن إضافتها .

(*r*) - صحيح .

٤٥

(*a*) المؤشر الرسمى ( dummy ) يجب أن يكون من نفس نوع البيانات التى سبق تعريفها .

(*b*) - صحيح .

(*c*) - صحيح .

(*d*) لايمكن تحديد عنصر من عناصر منظومة مضغوطة إلى إجراء أو إلى دالة كمؤشر فعلى .

(*e*) - صحيح .

(*f*) المؤشر الفعلى الثانى ( cost ) من نوع مختلف عن المؤشر الرسمى المناظر له .

(*g*) لايمكن إلا ضغط أخر بعد فقط لمؤشر منظومة تكوينية متعددة الأبعاد .

(*h*) - يجب أن يكون المؤشر الرسمى فى process مؤشر منظومة تكوينية متغيراً ، وليس مؤشر منظومة تكوينية قيمة .

(*i*) - المؤشرات الفعلية يجب أن تكون منظومات من النوع الحقيقى ، لها نفس الأبعاد .

## الفصل العاشر

٣٥

(*a*) - صحيح .

(*b*) - صحيح .

(*c*) - التعريف الصحيح لايحتوى على END . كما أن تعريف نوع المنظومة ( sales ) غير صحيح أيضا .

(*d*) - صحيح .

(*e*) - تحتوى هذه المشكلة على العديد من الأخطاء :

١ - لايمكن أن تظهر محتويات السجلات فى تعبير بوليان ( birthday = today ) .

٢ - الإشارة إلى year فى معيار التوقف ( UNTIL Year = 0 ) يجب أن يسبقها اسم السجل .

٣ - القيمة النهائية المتوقعة فى معيار التوقف ( Year = 0 ) تقع خارج المدى .

(*f*) - صحيح .

(*g*) - يجب أن يتبع الجزء المتغير من السجل الجزء الثابت ( أى أن quantity : integer فى غير موضعها ) .

(*h*) - الإشارات إلى أسماء حقول السجل يجب أن يسبقها اسم السجل ( employees [i]. maritalstatus أو employees [i]. divorced . child ) كما أن عبارة writeln غير صحيحة ، حيث إن divorced ليس حقلا متغيرا نشيطا .

(*i*) - معرف حقل tag-field لايمكن تمريره إلى إجراء كمؤشر متغير .

## الفصل الحادى عشر

٤٨ (*a*) - يجب أن تكون العبارة توضيحا متغيرا ، بدلا من تعريف للنوع .

(*b*) - صحيح .

(*c*) - يجب ألا تظهر FILE OF فى توضيح ملف النصوص .

(*d*) - صحيح .

(*e*) - مكون الملف لايمكن أن يكون ملفا آخرا .

(*f*) - التوضيح صحيح ، بالرغم من أنه غير ضرورى ، حيث إن output , input سبق توضيحهما كملفات نصوص .

(*g*) - الملف الذى سيكتب فى data لم توضع له قيم ابتدائية ( عن طريق rewrite ) . كما لايمكن استخدام writeln أيضا ، حيث إن data ليس ملف نصوص .

(*h*) - صحيح .

(*i*) - صحيح ، على أن يوجد توافق ( customer . custname = name ) قبل ظهور نهاية الملف .

(*j*) - لايمكن تمرير ملف إلى إجراء كمؤشر قيمة .

## الفصل الثانى عشر

٢٥

(*a*) [re,mi,fa,sol,la]
(*b*) [do,re,mi,fa,sol,la]
(*c*) [do]
(*d*) [re,mi,fa]
(*e*) [do,re,mi,fa]
(*f*) [ ]
(*g*) [ ]
(*h*) [do,re,mi]

٢٦

(*a*) [re,fa]
(*b*) [do,re,fa]
(*c*) [re,mi,fa,la,ti]
(*d*) [do,re,mi,sol,ti]
(*e*) [re,mi,fa,sol,la]

٢٧

(*a*) - خطأ .
(*b*) - صحيح .
(*c*) - خطأ .
(*d*) - صحيح .
(*e*) - خطأ .
(*f*) - صحيح .
(*g*) - صحيح .
(*h*) - صحيح .
(*i*) - خطأ .

(j) - صحيح .
(k) - صحيح .
(l) - خطأ .

٢٨
(a) - صحيح .
(b) - يجب أن يكتب تعبير بوليان على النحو التالى :

(c) - يجب أن تكتب عبارة التحديد على النحو التالى :

(d) - صحيح .
(e) - صحيح .
(f) - العنصر الأول فى تعبير بوليان لايمكن أن يكون متغيراً من نوع الفئة .

## الفصل الثالث عشر

٣٥

(a)
```
TYPE link = ↑customer;
     customer = RECORD
                    name : PACKED ARRAY [1..20] OF char;
                    acctno : 1..9999;
                    balance : real;
                    next : link
                END;
```
(b)
```
VAR nextcustomer,lastcustomer : link;
```
(c)
```
new(nextcustomer);
   .
   .
   .
dispose(nextcustomer);
```

٣٦

(a)
```
lastcustomer := nextcustomer↑.next;
```
(b)
```
nextcustomer↑.next := NIL;
```
(c)
```
lastcustomer↑ := nextcustomer↑;
```
(d)
```
IF nextcustomer↑.acctno = 1330 THEN
   nextcustomer↑.balance := 287.55;
```
or
```
WITH nextcustomer↑ DO
   IF acctno = 1330 THEN balance := 287.55;
```
(e)
```
PROCEDURE readinput (VAR customer : link);
VAR count : 1..20;
BEGIN
   .
   .
   count := 1;
   WHILE NOT eoln DO
 BEGIN
   read(customer↑.name);
   count := succ(count);
 END;
```

```
   .
   .
END:

BEGIN   (* main action block *)
   .
   .
   readinput(nextcustomer);
   .
   .
END.
```

(*f*)
```
WHILE nextcustomer <> NIL DO
   BEGIN
      .
      .
      .
   END;
```

٣٧ أ – تخطيط البرنامج يحتوى على ٣ أخطاء :

١ – لايتفق المعرف من نوع المشير مع اسم متغير الإشارة .

٢ – الحقل المشير داخل متغير الإشارة له تحديد نوع غير صحيح .

٣ – توضيحات المتغيرات لها تحديدات نوع غير صحيحة .

ب – تخطيط البرنامج يحتوى على خطأين :

١ – لايحتوى تعريف متغير الإشارة على مشير .

٢ – المؤشر الذى يظهر فى عبارة new غير صحيح .

ج – صحيح . ( ينتج هذا البرنامج قائمة بالأسماء ) .

رقم الإيداع ٩٧٢٠/ ١٩٩٠

مطابع المكتب المصرى الحديث
MODERN EGYPTIAN PRESS
صندوق بريد ٢٥ شبرا القاهرة - القاهرة ت : ٢٢١١٠٧١ - ٢٢١١٠٧٤ فاكس ٢٢١١٠٧٢

## صدر أيضا للناشر
## في
## الحاسبات




## الناشر

**الدار الدولية للنشر والتوزيع**

٣٨ ش الأهرام – روكسي – مصر الجديدة
ص . ب ٥٥٩٩ هليوبوليس غرب – القاهرة
ت : ٢٥٨٢٨٨٧
تلكس : ٢٠٠٧٠ PBCRB UN
فاكس : ٠٠٢٠٢ / ٢٩١٨٠٥٩

ISBN: 977-5107-05-9